作者简介

中国计量大学 Lqinyi　ECFinal亚洲区域总决赛铜牌

作者寄语

由于市面上的语言基础书籍都是偏向于大学生年龄段的，特此出此书籍帮助中小学生理解C语言，使用C语言。仅供童心智造OI使用！。

人一我十，人百我万，是我们能比上那些真正有天赋的人唯一途径，希望大家能够不懈的努力。

我们学习算法竞赛需要明确以下问题

1. 我为算法竞赛能做出什么牺牲?
2. 我是否有超越别人的想法?
3. 当我落后于别人时我是自暴自弃还是加强练习?
4. 我参与竞赛的主要目标是拿奖，还是提升自己?
5. 时刻常问自己，努力了没有拿奖是否会后悔?
6. 在学习算法竞赛的路上能否做到常怀感恩之心，常怀学习之心?
7. 能否做到一直相信自己，相信老师，相信父母?
8. 能否做到相信题目都是人出的，是人能写的，是自己可以完成的?

最后引用——陈立杰《2013成都区域赛开幕式讲话》

> 我的大学时光纯粹
> 那时我还年轻，也想不到以后会怎么样，就是想要呆在机房里，为了做出题目这样纯粹的感动而活下去。一晃五年过去了，从我AC第一道A+B以来，世界已经变了太多，曾经的感动和梦想似乎也随风而去。然而我心中却始终回荡着几天后AC的欢呼声，差几秒钟提交的捶桌声，比赛失利的叹息声，以及永恒的机房主题----键盘声。我还记得很久以前有人跟我说的话，自己选择的路，跪着也要走完。朋友们，虽然这个世界日益浮躁起来，只要能够为了当时纯粹的梦想和感动坚持努力下去，不管其它人怎么样，我们也能够保持自己的本色走下去。

目录

1. Hello C语言

平常随处可见的电脑想要运行起来，软件运行起来，需要我们人类告诉电脑如何运行，但是电脑又听不懂我们的话那该怎么办。这就创造了我们的语言: C语言，C++，JAVA，Python。但是我们语言这么多种我们为什么就偏要学C语言，C++呢?

优点：快。缺点：难。

那么C语言和C++有什么区别呢，具体来说C语言是C++的爹，因为C++是在C语言的基础上进行改进的。

所以C语言能用的C++也能用，C++能用的C语言用不了，所以我们可以先学C语言再学C++。

下面一起来看一段代码:

```
#include<stdio.h> //include在英文是包含的意思，包含stdio.h里面的一些内容(C语言作者)

int main(){ //告诉电脑我们从这里开始运行

    printf("Hello C语言"); //输出一句话 Hello C语言
    printf("Hello C语言2"); //输出一句话 Hello C语言2
    printf("Hello C语言3"); //输出一句话 Hello C语言3
    printf("Hello C语言4"); //输出一句话 Hello C语言4
    //printf("Hello C语言5");
    return 0; //返回0
}
```

我们一句一句来解析到底干了什么事情

1. 首先就是 `#include<stdio.h>`

   include在英文是包含的意思，包含stdio.h里面的一些内容

   这些内容是由创造C语言的人给我们的工具箱，就像下面的 printf就是工具箱里面的一个工具

2. 其次就是main函数

   他告诉电脑，从他这里开始运行代码，代码必须从上往下一句一句执行。(程序的入口)

3. {} 他里面写我们想要干的事情，包含内容。

4. printf 前面提到了它是一个工具，这个工具是用来输出的，可以输出中文，字母，数字等

5. return 0

   return是返回的意思，这句话就是告诉电脑，我做完了这件事得到了什么结果

6. "" 双引号，里面的内容就是我们想要输出的东西

7. ;

   这个标识相当于语文中的句号，表示我们说完了这句话，称作语句
   只有计算机看得懂的合法操作才算是语句

8. // 注释，把一些不需要机器运行的句子给注释掉。

1.1 练习:

1. 以下哪些是合法语句?

```
1;
asd;
"111";
231;
();
*;
&;
^;
'111';
'&';
"&^*%$";
;
```

2. 试验以下操作
   1.如果写两个main函数会怎么样?
   2.如果不写return会怎么样
   3.如果写return 1，return 2会怎么样
   4.如果不写#include<stdio.h>会怎么样
   5.如果printf 不写""会怎么样

2. 格式化输出

2.1 示例程序

```
#include<stdio.h>
int main(){
    printf("我是大帅哥");
    printf("我是大美女\n");
    printf("我是小帅哥");
    printf(" 我是小美女");
    return 0;
}
```

大家通过示例程序会发现最后输出如下

```
我是大帅哥我是大美女
我是小帅哥 我是小美女
```

发现printf("")双引号里面写了什么他就会输出什么，但是我们主观的去给他换行他是没有换行成功的，就如我是大帅哥，我是大美女这两个输出是在一行，但是我是小帅哥又是在下一行，说明我们的 **\n**起了作用，就是换行的意思，那么空格的话，也是需要我们自己手动去输出才可以的。

2.2 错误案例分析

很多同学一开始学的时候会对代码概念模糊，但是没有关系，老师带你们认识一下一些正常错误

1. 漏掉；

```
printf("123")
```

2. printf里面写了双引号的缺失

```
printf(123);
printf("123);
printf(123");
```

3. 英文字符写出中文字符

```
；  ；
” ""
‘ '
《 <
# #
（ (
//前者是中文字符后者是英文字符，是不一样的
```

4. 写的时候，多个双引号或者使用逗号

```
printf("123""123"); //虽然是可以的，但是让人误解
printf("456","123"); //只输出456 不输出123，因为printf只输出第一个字符串
```

2.3 练习:

1. 请问以下代码是否会产生错误？若不产生错误会输出什么结果

```
printf("");
```

```
printf('%c','a');
```

```
printf("123""123");
```

```
printf("123","456", "789");
```

```
printf("Baa Baa Black Sheep.");
```

```
printf("Baa\n Baa\n Black\n Sheep.");
```

2. 以下哪个选项能在电脑上输出( )(多选)

```
  *
 ***
*****
 ***
  *
```

A. $printf(" \ * \ ");printf(" \ * ** \ ");printf(" \ * ** ** \ ");printf(" \ * ** \ ");printf(" \ * \ ");$

B.
$printf(" \ * \backslash n \ ");printf(" \ * ** \backslash n \ ");printf(" \ * ** ** \backslash n \ ");printf(" \ * ** \backslash n \ ");printf(" \ * \backslash n \ ");$

C.$printf(" \ * \backslash n \ * ** \backslash n \ * ** ** \backslash n \ * ** \backslash n \ * \backslash n \ ");$

3. 变量

在计算机中，分了好多种数据，一种是整数，一种是浮点数，一种是字符，还有布尔
举个例子：生活中有很多东西，比如书，我们想要把书存起来就要放到书架上去。
变量就像做菜一样，我们要把各种各样的调料分类，放到一个容器里面。
计算机也是这样，要把各种各样的数据放到不同的容器里面，那么变量就是容器
那么我们想要在计算机中存我们的数字，整形。那么我们必须要向计算机的内存申请一块空间；

3.1 示例程序

```
#include<stdio.h>

int main(){
    int a = 1;
    int b = 2;
    printf("第一个数%d，第二个数%d",a,b);
    return 0;
}
```

我们发现最后程序的输出如下

```
第一个数1，第二个数2
```

说明我们利用a和b来存储了数字并且成功的把他输出出来了，通过（%d）占位符 来表示输出的位置，用a来表示输出的值。

3.2 变量，赋值

3.2.1 变量的数据类型

常见的变量数据类型有以下几种:

变量书写方式	变量代表的类型	占位符
int	整形	%d
long long	长整形	%lld
float	单浮点数	%f
double	双浮点数	%lf
bool	布尔	无
char	字符	%c

整形，长整型是用来存储整数的，单浮点数和双浮点数是用来存储有小数的数，布尔用来存储真假，字符用来存储字符。

当我们想要输出我们特定的变量的时候，我们就必须使用相应的占位符去占住位置

```
#include<stdio.h>

int main(){
    int a = 1;
    long long b = 100;
    float c = 1.0;
    double d = 2.0;
    char e = 'A';
    printf("%d %lld %f %lf %c",a,b,c,d,e);
    //这个读作分别以整形类型，长整型类型，单浮点数类型，双浮点数类型，字符类型去输出a，b，c，d，e；
    return 0; //返回0
}
```

那么大家可以看到，字符是用 ' 单引号引起来的，记住字符只能是单个的，不能是多个的，字符串才可以是多个字符(双引号就是字符串)。

3.2.2 变量的声明

```
int a;
```

这行代码就叫做**变量的声明**，声明是C语言中最重要的特性之一。在该例子中，声明完成了两件事情：
其一，在main里面有一个名为a的变量。**其二**，int 表明a是一个整数。
int 是C语言中的关键词，所以不能使用int作为变量名或者其他作用。
a是一个标识符，也就是一个变量，函数或者其他实体的名字。
在C语言中所有的变量使用，**必须先声明再使用。**
具体形式例如 int a;（读作整形变量a） char a（读作字符变量a）; long long a（读作长整形变量a）；
取名也是有规则的: 不能以数字开头，不能起重名。
可以用小写字母，大写字母，数字和下划线(_)来命名。而且名称的第1个字符必须是字母或下划线，不能是数字。

有效的名称	无效的名称
wiggles	$Z]**
cat2	2cat
Hot_Tub	Hot-Tub
_kcab	Tax rate

注意1：可以同时声明多个标识符

```
int a,b,c; //这样是被允许的，但是需要用逗号隔开名字，同时将标识符a，b，c都声明整形变量
```

注意2: 倘若你声明了一个变量，此时你并不知道变量里面是什么所以不要乱用。

3.2.3 赋值

```
int a = 1;
```

大家可以看到 = 就是我们的赋值，把1赋值给了整形变量a，那么赋值不管在什么时候进行都可以，可以在声明的时候进行也可以在声明后进行，也可以在赋值完再赋值，我们把 = 号**左边的称为左值**，**右边称为右值。**

```
int a = 1;
int b;
b = 1;
b = 2;
int c,d = 1,f = 2;//以上统统合法 但是注意，如果对一个标识符多次赋值的话，每一次赋值都会覆盖掉上一次的值，此时b里面的数是2而不是1
```

赋值我们还需要注意的如下：

```
int a = 1;
int b = 2;
int c = a = b = 4;
//此时c，a，b的数据都为4
//因为先运算 b = 4,再运算a = b，再运算c = a。
//所以赋值运算时是从右往左看
```

```
int b = 2;
1 = b; //这样是错误的因为常量不能当作左值
```

```
长整型，整形变量赋值如下 //可以先定义再赋值，也可以一边定义一边赋值
int a = 1213;
int b; b = 1231;
long long c = 123123;
long long d; d = 123123;
字符变量赋值如下
char a = 'a';
char b; b = '1'; //必须使用单引号，单引号代表字符
浮点数变量复制如下
float a = 123.123;
float b; b = 123; //浮点数可以存储整数，整形不能存储浮点数
double c = 123123;
double d; d = 123.12321;
布尔变量赋值如下 //只有两种状态，真假
bool e = true;
bool f; f = false;
```

3.2.4 输出变量的值

输出变量的值需要我们通过占位符去输出，一个占位符对应一个标识符，因为我们不能同时在同一个地方输出多个(没有意义);

```
int a = 1,b = 2,c = 3;
printf("%d %d %d",a,b,c);
//输出1 2 3，第一个占位符对应a，第二个占位符对应b，第三个对应c
double a = 12.3333,float b = 123.566666;
printf("%.2lf %.3f",a,b);
//输出 12.33 123.566
//.几就是保留几位小数输出
```

3.3 错误案例分析

3.3.1 关于变量的使用

使用变量最典型的错误就是大家会忌惮他是自己取的名字，不能把他当成已知数来使用，其实大家可以就把他当成数字来使用，所以数字不合法的操作变量就是不合法。

```
int a = 1;
printf("%d %d",a,3);
//很多同学认为以上例子就是错的，但是是对的，因为他无非就是把3按照整形类型输出，3本身就是整数所以肯定是对的。
int a = 1;
printf("%d %d",-a,-3);
//有些同学认为以上例子也是错的，但是变量其实就是已知数，我们可以把上面成 printf("%d %d",-1,-3);
int a = 1;
printf("%d %d %d",a,a,a);
//有些同学认为以上是错的，但是是可以的，无非就是在三个地方输出a的值
int a = 1,b = 2;
a = b;
//有些同学认为以上是错的，但是是可以的。
int a;
printf("%d",a);
//有些同学定义了后直接输出，会产生意想不到的值，所以我们必须自己给他赋值后才能使用。
```

3.3.2 关于赋值的使用

同学们对于赋值的使用唯一可能产生的错误可能就是如下

```
char a = "123";
//字符赋值为字符串,这是错误的
char a = '123';
//当字符里面有多个字符时不会产生错误，但是a里面最后会等于3也就是最后一个字符。
int a = 123.345;
//当整数等于浮点数的时候，浮点数的小数点自动消失，也就是a = 123;
float a = 123;
//当浮点数等于数字的时候，会自动填上小数点
```

3.4 练习:

1. 请问以下哪些操作是合法的;

```
int a = 1;
a = -a;
```

```
char a = 'a';
```

```
char a = '1';
```

```
char a = '123';
```

```
char a = "1";
```

```
float a = 1;
```

```
double a = 0
```

```
int a;
a = 1;
```

```
printf("%d",1);
```

```
int a = 1;
printf("%d",&a);
```

```
printf("12314%d",1);
```

```
printf("%d",1,2,3);
```

```
printf("%d %d %d",1);
```

```
int a = 1,b,c,d = 1;
```

```
int a,int b,int c;
```

```
int a;
printf("%d",a);
```

2. 下面是林老师写的一个程序想要问一下你的意见有没有错误

```
#include<stdio

int main(){

    int a,b,c = 2;
    printf("%d",a,b);
    printf("%d %d",c)
    return;
}
```

3. 请问以下代码会输出什么?

```
int num;
num = 2;
num = 3;
printf("%d + %d = %d",num,num,num + num);
```

```
printf("123","1234");
```

```
printf("123123""123");
```

```
printf("123%d""123%d",4,5);
```

```
printf("%d%d",1,2);
```

```
int a = 100;
int b = a = 12 * 10;
```

4. 考虑以下程序

```
#include<stdio.h>
int main(){
    int a,b;
    a = 5;
    b = 2;//第7行
    b = a;//第8行
    a = b;//第9行
    printf("%d %d",b,a);
    return 0;
}
//请问执行完第7，8，9行后程序的状态分别是什么 (a等于几，b等于几)
```

4. 格式化输入

有输出，那么肯定会有输入，输入通过我们的键盘输入
那么输入的东西想要保存的话，一定需要我们的变量去存储！

4.1 示例程序

```
#include<stdio.h>

int main(){
    int a = 1,b;
    float c;
    scanf("%d %d %f",&a,&b,&c); //就是我们的输入函数
    printf("%d %d %f",a,b,c);
    return 0;
}
```

以上程序，当我们输入 3 4 5.2的时候，我们把3放进了a，4放进了b，c放进了5.2会输出 3 4 5.2

其实scanf和我们的printf有点类似

相似点：

1.都是需要占位符，因为我们也要把特定的数据放到特定的变量里面

2.都是按照特定的格式输入(输出)

```
#include<stdio.h>

int main(){
    int a = 1,b;
    scanf("%d.%d",&a,&b);
    printf("%d.%d",a,b);
    return 0;
}
```

以上程序当我们输入 123.25时 输出了 123.25

```
#include<stdio.h>

int main(){
    int a = 1,b;
    scanf("%d %d",&a,&b);
    printf("%d.%d",a,b);
    return 0;
}
```

以上程序当我们输入 123.25时 输出的不是 123.25，说明我们在scanf的时候加上了个 . 他就会当作一个字符给读取了

不同点：

一个加了取地址 & 一个不需要加取地址

4.2 形象化

举个例子char a,b,c,d;
scanf("%c%c%c.%c",&a,&b,&c,&d);
那么对于计算机来说的话，我们的scanf其实相当于一个原料分拣窗口，%c相当于一个接受字符类型的占位符，相当于分拣器的一个框框。
当我们的计算机如果期望读到 “[一个字符] [一个字符] [一个字符] . [一个字符]” 按照这样子格式输入的数据。

当读到123.4的时候他就会把这个读入拆分成"['1'] ['2'] ['3'] . ['4']"，当每个框框都读取到足够的数据后，就会让后面的框框读取
第一个分拣框把 数据 '1'转交给了变量a，一直到第四个分拣框把数据'4'交给了变量d。 那么转交给他的话就是需要我们的地址符号&
所以为什么要把scanf说成格式化输入，因为必须要按照我们写的格式进行一个输入才可以。
那么 为什么输入的时候要加 & 取地址符号，为什么输出的时候不需要加呢?
我们考虑一个例子:
我们快递员送快递的时候，知道是我的快递就能送到我家？肯定是需要知道我家的地址才可以送到。
所以我们的输入必须要知道变量在空间的位置才可以把数据放进去。
那么输出的时候为什么不需要加?
考虑一个例子:
我们快递到了之后，我们只需要回想一下，之前买了什么，就能猜个大概是什么，我们的电脑脑子比我们好用多了，所以一定能回想起来。

4.3 错误案例分析

同学们一开始对于scanf不熟悉会产生以下问题

```
int a;
scanf("%d",&a,&a);
//以上代码是错的，因为一个占位符，必须对应一个变量名
int a,b;
scanf("%d %d",&a);
//以上代码也是错的
int a;
scanf("123%d",a);
//当输入1234的时候，a = 4，因为123被吃掉了
int a,b;
scanf("%d %d",&a,&b);
//当输入12的时候，不会把1给a，2给b。因为a是按照整形输入，12是连续的数字，所以12都会给a。
int a,b;
scanf("%d%d",&a,&b);
//当输入12 345时，a=12，b=345。计算机会自动识别空格不是数字。
```

4.4 练习:

1. 输入一个整形 a 以下哪个选项正确（ ）

   A.scanf("%d",&a);　B.scanf("%c",&a);　C.scanf("%d",a);　D.scanf("%f",a);

2. 输出一个双浮点数 a 以下哪个选项正确（ ）

   A.printf("%d",a);　B.printf("%c",&a);　C.printf("%f",&a);　D.printf("%lf",a);

3. 输出我是大傻蛋 以下哪个选项正确（ ）

   A.printf("我是大傻蛋");　B.printf(我是大傻蛋);　C.printf("我是大傻蛋")

4. 请问以下哪些操作是合法的

   ```
   int a;
   scanf("%f",&a);
   ```

   ```
   int a;
   scanf("%d",a);
   ```

   ```
   int a = 2;
   scanf("%d",&a);
   ```

```
int a;
scanf("%d %d %d",&a,&a,&a);
```

```
float a;
scanf("%c %d %f",&a,&a,&a);
```

5. 请思考什么是空白？什么是空格？

5. 字符串+转义符+注释

字符串: 用双引号来 引起的一串字符称作 字符串字面值 简称 字符串
例如 "123asdas" ， "1231￥#￥%12"
如果有一个字符串就是为 ""里面什么都没有，我们称作空字符串
像我们的scanf 和 printf里面的就是字符串

转义符: 转义字符就是为了使用无法被设备表示的操作，例如我们的换行操作，我们不知道如何去表达，所以产生了转义符。

字符表

所有的转义字符和所对应的意义：

转义字符	意义	ASCII码值（十进制）
\a	响铃(BEL)	007
\b	退格(BS)，将当前位置移到前一列	008
\f	换页(FF)，将当前位置移到下页开头	012
\n	换行(LF)，将当前位置移到下一行开头	010
\r	回车(CR)，将当前位置移到本行开头	013
\t	水平制表(HT) （跳到下一个TAB位置）	009
\v	垂直制表(VT)	011
\\	代表一个反斜线字符"\'	092
\'	代表一个单引号（撇号）字符	039
\"	代表一个双引号字符	034
\?	代表一个问号	063
\0	空字符(NUL)	000
\ddd	1到3位八进制数所代表的任意字符	三位八进制
\xhh	十六进制所代表的任意字符	十六进制

注释：注释有两种，一种是 // ，另一种是 /* */

6. 数组

6.1 示例程序

```
#include<stdio.h>

int a[10];
int main(){

    scanf("%d %d %d %d,&a[0],&a[1],&a[2],&a[3]);
    printf("%d %d %d %d",a[0],a[1],a[2],a[3]);
    return 0; //返回0
}
```

以上程序当我们输入 1 2 3 4的时候，我们发现输出的也是 1 2 3 4，我们发现用一个声明就定义了好多个标识符。

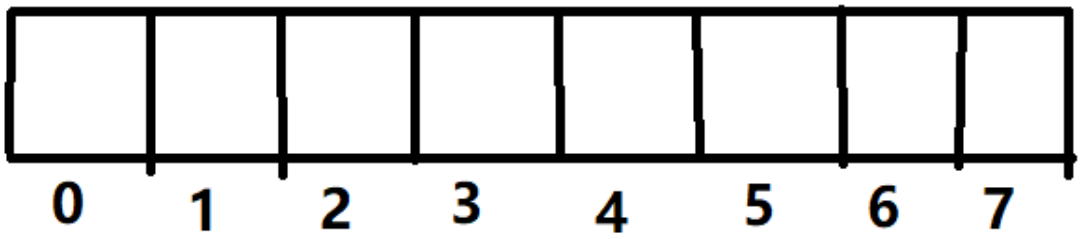
如果数据多了的话其实我们取名字是件很麻烦的事情

就像我们人类取名字就是件麻烦的事情，还好有姓氏已经确定了一个字

所以C语言也考虑到了这件事，对于同一个变量类型的，我们用数字表示他不就行了吗，但是数字又不能用来起名，所以C语言作者就创造了数组这个东西 例如 int a[10]，这个读作a是一个大小为10的数组 或者读作 大小为10的整形数组

这个东西就好比我们的商品房，第一层住的是小红家，第二层住的是小刚家....

只不过数组第一层用0表示，大小为8的数组最后一层是7。



那么正式的说，数组就是同类型数据元素的有序序列。

6.2 错误案例分析

很多同学在见到数组的时候，和第一次见到变量一样感觉和我们的数字，字符有区别，但是数组也是已知数，我们把他当已知数使用

```
int a[10],b[10];
scanf("%d %d",&a[0],&b[1]);
a[0] = -a[0];
b[1] = a[0] + 23;
//以上是正确的，可以看到数组的用法和我们的变量是一样的，只不过多了一个大括号一个数字而已
int a[10];
scanf("%d",&a[10]);
//以上是错误的，因为我们能使用的是从编号0开始也就是a[0],最多用到a[9],那么a[-1],a[10]，我们都是不可以去使用的
```

6.3 练习：

1. 一个大小为100的整形数组 如果我们想要使用的话，以下哪个选项合法( ) （多选）

   A.a[0]　　B.a[-1]　　C.a[99]　　D.a[100]

2. 想要给大小为3的整形数组中每个位置都输入一个数，以下哪个选项正确( )

   A.scanf("%d %d %d",&a[0],&a[1],&a[2])

B.scanf("%d %d %d",&a[1],&a[2],&a[3])

C.scanf("%d %d %d",a[0],a[1],a[2])

D.scanf("%d %d %d",a[1],a[2],a[3])

3. char a[100]，float b[10000]，char c[1123] 分别怎么念

4. 想要把大小为3的整形数组中存的每个数都输出出来，以下哪个选项正确( )

   A.printf("%d %d %d",&a[0],&a[1],&a[2])

   B.printf("%d %d %d",&a[1],&a[2],&a[3])

   C.printf("%d %d %d",a[0],a[1],a[2])

   D.printf("%d %d %d",a[1],a[2],a[3])

5. int a[10]数组里面的每个元素是什么？每个元素的属性是否一样？

6. 请用代码写出以下题目

   - 输入5个整数(不超过整形范围)，输出 第2个输入的，第3个输入的。

7. 请问以下操作是否合法

```
int a[10];
scanf("%d",&a[11]);
```

```
int a[100];
scanf("%d %d %d",&a[12],&a[12],&a[99]);
```

```
int a[10];
a[1] = 1;
a[2] = 2;
scanf("%d %d",&a[3],&a[2]);
printf("%d %d",a[2],a[3]);
```

```
float a[10],b[10];
a[1] = 1;
a[2] = 2;
a[3] = a[4] = a[5] = 7;
```

```
int a[10],b[10],float d[10];
```

7. + - * / %原码补码反码

7.1 基本运算符 + - * / %

C语言中的 + 就是数学中的 +

C语言中的 - 就是数学中的 -

C语言中的 * 就是数学中的 ×

C语言中的 / (整除)和数学中的 ÷(除) 不太一样，数学中 1 ÷ 2 = 0.5 但是C语言是分为整形，浮点数的，所以C语言中 整形/整形=整形(向0取整) 浮点数/整形=浮点数，浮点数/浮点数=浮点数 整形/浮点数=浮点数
什么叫向0取整呢举两个例子 1 / 2 = 0，-1 / 2 = 0。 我们可以把 -1 / 2 看成 -(1 / 2)，1 / 2 = 0，所以答案 -(1 / 2) = 0。这就是向0取整

%叫做取模运算，也叫做整数取余。例如 a % b叫做a对于b取模。
在现实数学中 被除数 / 除数 = 商 .....余数。那么被模数 % 模数 = 余数，商我们就过滤掉了

```
3 % 1 = 0;
5 % 2 = 1;
4 % 2 = 0;
```

余数 = 被除数 - 商 * 除数，商 = 被除数 / 除数 所以 余数 = 被除数 - 被除数 / 除数 * 除数

用a代表被除数，b代表除数就是

a % b = a - a / b * b （向下取整) **注意：**取模只能整数对整数，不能其他类型！，并且取模也不能对0取模 // 参考除法

除此之外 + -，也可以用来代表正数负数

```
变量也可以直接增加 + -
int a = 1;
printf("%d",-a); //能够成功输出-
```

7.2 示例程序

```
#include<stdio.h>

int main(){
    printf("%f %f %f %f\n",3/2,2/2,-1/2,-2/2);
    printf("%f %f %f %f\n",1.0/2,1/2.0,-1.0/2,-1.0/-2.0);
    return 0; //返回0
}
```

可以看到我把所有运算都按照浮点数输出，但是第一行输出0.000000 0.000000 0.000000 0.000000，第二行输出0.500000 0.500000 -0.500000 0.500000
说明第一行的运算，运算答案不是浮点数，所以才会出现都是0的情况，如果我们改成 %d那么会发现 输出1 1 0 -1。
所以整形之间运算答案是整形，所以我们同学在写题的时候要注重一下变量。
那么除此之外，计算机中的运算当然可以是变量之间的运算，并且有运算优先级(* / %优先)

```
#include<stdio.h>

int a,b,c;
double d;
int main(){
    a = 10,b = 100,c = 10,d = 1.0;
    printf("%d %d %d %d %d %lf\n",a * b,a + b,c - b,(a + b) * c,a + b * c,d / a);
    return 0; //返回0
}
//输出  1000 110 0 1100 1100 0.1000000
```

注意：从加减乘法我们可以体会出变量之间的不同了，首先就是整形和长整型的不同，如下

```
//我们对于以下代码进行试验
#include<stdio.h>

int a,b,c;
int main(){
    a = 100000000,b = 100000000000,c = 100;
    printf("%d %d %d\n",a * b,a + b,c - b);
    return 0; //返回0
}
```

我们发现与我们的预期值不一样，为什么相乘得到了负数，相加变小了，相减小了这么多。

这个时候就要说到我们的存储大小了。

7.3 存储

给大家举个例子，我们现实中的盒子有大有小，大的装的东西比较多，小的装的东西比较小，如果我们用大的容器装很少的东西我们会觉得很浪费，如果用小容器装很大的东西又装不下，计算机也是一样，它能装的东西有限，所以C语言作者给每个变量都设置了一定的大小。

int	整形	4个字节
long long	长整形	8个字节
float	单浮点数	4个字节
double	双浮点数	8个字节
bool	布尔	1个字节
char	字符	1个字节

所以若我们改成

```
//我们对于以下代码进行试验
#include<stdio.h>

long long a,b,c;
int main(){
    a = 100000000,b = 100000000000,c = 100;
    printf("%lld %lld %lld\n",a * b,a + b,c - b);
    return 0; //返回0
}
```

运算得结果就正确了

7.4 练习:

1. 请问如何用代码实现以下问题
   输入一个整数a，若a是奇数则输出1否则输出0
   输入两个整数a，b输出 a * b
   输出 整形 a 对 整形 b取模后的答案
2. 请问以下代码合法吗?

   ```
   printf("%d",1 + 2 + 3);
   ```

   ```
   printf("%d %d",1 + 2,1 * 2);
   ```

```
int a = 1,b = 2;
printf("%d",a + 2 * b);
```

```
printf("%d",1,1);
```

```
printf("%d",1 + -1);
```

```
printf("%d",1 - +1);
```

7.5 前言

要学习以下知识，我们考虑两个数学问题

1. $\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+....+\frac{1}{n}+\frac{1}{n*2}$
   思考这个题我们可以给他转换成给以上数字再添上一个 $\frac{1}{n*2}$
   那么答案就变成 1 - $\frac{1}{n*2}$
2. 通过以上例子思考 1 + 2 + 4 + ... + n + n * 2 = ?

7.6 原码反码补码

那么到底什么是字节呢，字节就是最小的单位吗?

其实计算机中最小的单位是位。为什么是位? 答：首先位是对于二进制来说的，我们平常的数字是十进制，十进制是0~9后变成10就需要进位，我们分解十进制看看 对于 123456 这个数来说其实是 $1 * 10^5 + 2 * 10^4 + 3 * 10^3 + 4 * 10^2 + 5 * 10^1 + 6 * 10^0$，那么对于 二进制来说 11110来说分解就变成 $1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1* 2^1 + 0 * 2^0$。我们发现二进制每一位只有两种状态，0和1，跟我们人类平常所说的 有或者没有，真或者假很像，表达的很方便，所以为了减少计算机的复杂程度，为了遍历，我们使用了二进制的 **位** 作为最小单位。
1字节(Byte) = 8位/比特(bit)

7.6.1 原码

那么对于 int类型来说他是 4字节 也就是32位，它最多只能装32位数所以最大也只能装$2^0 + 2^1 + ... + 2^{31}$，**但是这样子的话不能表示负数了，所以我们把他的第一位设置为符号位，用符号位来表示这个数是整数还是负数，如果符号位为1的话他就是负数，如果符号位为0的话他就是正数，因为-1^0 = 1,-1^1 = -1，这样的表达方式称作原码。**
一般表达形式为

```
(100101)原 = -5
(010010)原 = 18
当然对于-5这个数有很多表达形式 可以为 1101也可以为10101，只要正确即可
```

但是我们发现如果用电路去表示的话会显得特别复杂，因为对于一个运算 1 - 1 = 0来说我们可以表示成 1 + (-1) = 0，用原码来算的话就是 00000001 + 10000001 = 10000010 = -2，我们发现运算错误，所以不得不多加些判断。

7.6.2 反码

为了解决这个问题我们的反码就产生了。正数的原码=反码，负数的原码=反码符号位不变其他位取反，此时我们做运算我们就发现 得到的是我们想要的数 例如 1 + -1 用反码来算就是 (00000001)反 + (11111110)反 = (00000000)反 = (00000000)原 = 0。

一般表达形式为

```
(100101)原 = (111010)反 = -5
(010010)原 = (010010)反 = 18
```

但是我们又发现了个问题，10000000 和 00000000 是+0 和 -0都是0是没有意义的，浪费了一个数。

7.6.3 补码

为了解决这个问题补码就出现了，我们把符号位直接去掉，最高位代表-2^k那么 1011 表示的就是 1 * -2^3+0 * 2^2 + 1 * 2^1 + 1 * 2^0。
最终除了最高位为负数，其他都为负数，所以取范围最小值只需要取最高位，范围最小值只需要取去掉最高位即可。

一般表示形式为

```
(100101)补 = -27
(010010)原 = 18
```

所以我们int的数据范围就可以通过补码算出来

最小范围就是 -2^{31}，最大范围 = 2^0+2^1+....+2^{30} = 2^{31}-1大约是 2e9左右

那么通过以上分析，我们就大概知道为什么乘法加法减法得不到我们的预期值了，原来是溢出了，水装不下满出来了，计算机装不下就给他删掉了。

那么除此之外补码也可以由原码反码转换过来。

首先大家可以看到，当十进制为正数的时候，所有码第一位总是为0，所以就和普通二进制没区别

当十进制为负数的时候，假设现在有4位。那么4位二进制的范围为0 ~ 15。当原码表示-x的时候，是通过后3位表示x，反码就是后三位取反，原码是要把第一位取做1，所以一定有一个 -2^3 + y = -x 那么，y = 2^3 - x，反码的后三位取反相当于 2^3-1-x，所以补码 = 反码 + 1。

7.7 总结：

计算机做的运算都是以补码形式进行运算

正数 : 原码 = 反码 = 补码

负数 : 原码 ， 反码 = 原码符号位不变其他位取反，补码 = 反码 + 1

每一个类型的变量都是由一定的数据范围以及内存的

7.8 练习

1. C语言中运行的机器码是原码还是补码还是反码?
2. 以下哪个选项会溢出（） （多选）

   A. int a = 10000000000;　　B.char a = 256　C.long long a = 1000000000

   D.long long a; int b = 10000000;int c = 1000000000; a = b * c;

   E.long long a; long long b = 10000000;int c = 1000000000; a = b * c;

   F.long long a; int b = 10000000;long long c = 1000000000; a = b * c;
3. 十进制10213转换成二进制是多少
4. 十进制3214转换成二进制是多少
5. 二进制1110011转换成十进制是多少
6. $(11100101)_{原码}$转换成反码和补码是什么
7. $(11100101)_{反码}$转换成原码和补码是什么

8. $(11100101)_{补码}$转换成原码和反码是什么

9. 算出bool，char，long long的数据范围

10. 请思考，如何直接判断原码，反码，补码是否为负数

8. 存储

上一章我们分析了计算机的存储是通过二进制的补码来存储的，那么我们的字符到底是怎么存储的？如果所有数据之间都是二进制补码，那么字符和数字之间可以相互转换吗？浮点数和整数之间可以互相转换吗？

8.1 ASCII

上面题到存储数据是使用二进制的补码那么英文字母，奇奇怪怪的字符，中文都是怎么存的?

为了解决这个问题，人们发明了ASCII表，当我们使用字符的时候C语言自动发生翻译产生我们想要的东西。

ASCII 字符代码表 一

高四位 / 低四位		ASCII非打印控制字符										ASCII 打印字符													
		0000					0001					0010		0011		0100		0101		0110		0111			
		0					1					2		3		4		5		6		7			
		十进制	字符	ctrl	代码	字符解释	十进制	字符	ctrl	代码	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	ctrl
0000	0	0	BLANK NULL	^@	NUL	空	16	▶	^P	DLE	数据链路转意	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH	头标开始	17	◀	^Q	DC1	设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☻	^B	STX	正文开始	18	↕	^R	DC2	设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX	正文结束	19	‼	^S	DC3	设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	♦	^D	EOT	传输结束	20	¶	^T	DC4	设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ	查询	21	§	^U	NAK	反确认	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK	确认	22	■	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	震铃	23	↨	^W	ETB	传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◘	^H	BS	退格	24	↑	^X	CAN	取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	TAB	水平制表符	25	↓	^Y	EM	媒体结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◙	^J	LF	换行/新行	26	→	^Z	SUB	替换	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	竖直制表符	27	←	^[	ESC	转意	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	换页/新页	28	∟	^\	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|	
1101	D	13	♪	^M	CR	回车	29	↔	^]	GS	组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	♫	^N	SO	移出	30	▲	^6	RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	☼	^O	SI	移入	31	▼	^-	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ	^Back space

注：表中的ASCII字符可以用:ALT + "小键盘上的数字键" 输入

我们只需要记住A-Z的ascii码是 65~90 ， a - z的ascii码是97-122就行
通过这张表，我们发现每个字符和数字都是一一对应的

8.2 强制转换

既然我们的数据都是用二进制存的，只不过翻译的形式不一样，所以我们可以把它强制转换成其他类型

就像int a = 65；他如果强制转换成char的话 就会变成A 因为通过二进制来看他们两个是一样的

强制转换怎么转换呢，只需要在那个数前面打上 **(转换的类型)转换的变量** 或者 **转换类型(转换的变量)** 即可

8.2.1 示例代码

```
#include<stdio.h>

int main(){
    int a = 65;
    char b = (char)a;
    printf("%d %c ",a,b);
    char c = char(a);
    printf("%d %c",a,b);
    return 0;
}
//答案输出65 A 65 A
//同时我们也发现强制转换的时候他的类型只是当前改变，不是一直改变
```

但是强制转换有个弊端，那就是存储，**我们char的存储只有1字节，int的存储有4字节，所以当int强制转换成char的时候会有3个字节存不下。**

这个时候这3个字节就会被C语言丢掉(相当于溢出)，所以不能乱转换。

8.2.2 疑问点

之前讲过我们的printf，scanf是按格式输入和输出

```
int a = 65;
printf("%c",a);
char b = 'A';
printf("%d",b);
```

分析以上代码，我们按照字符格式输出整形a，按照整形格式输出字符b。
最后发现答案是 A 和 65，说明我们按照什么格式输出，他就会自动转变，所以有时候不需要强制换行。

8.3 练习：

1. 字符存储的形式是什么？是利用什么存储的？

2. 以下代码会输出什么？

```
int a = 120;
char b = (char)a;
printf("%d",(int)b);
```

```
int a = 1200;
char b = (char)a;
printf("%d",(int)b);
```

```
int a = 65;
printf("%c",a);
```

```
char a = 'A';
printf("%d",a);
```

```
char a;
scanf("%d",&a);//输入65
printf("%c",a);
```

---

9. 分支语句，关系运算符

9.1 分支1

前面学了取模%，但是在计算机中我们想要判断输入的数是奇数还是偶数怎么办

那么就需要一个判断语句，像我们平常时候说的如果，不然，否则。

在C语言中这些语句就是

```
if(){    //如果什么什么为真，执行大括号里面的语句 {}包含的意思

}else{   //否则

}
```

```
if(){  //如果什么什么为真，执行大括号里面的语句

}else if(){   //否则如果

}else{   //否则

}
```

那么在计算机中什么为真什么为假呢?
好家伙真假又是两种状态！所以计算机贴心的把1当作真，0当作假。
但是其他数字又算什么呢？所以计算机设定非0为真，但是真只能为1
所以对于分支语句我们可以这么写

```
if(1){
    printf("YES");
}
//输出YES
if(0){
    printf("YES");
}else{
    printf("NO");
}
//输出NO
if(0){
    printf("YES");
}else if(123){
    printf("NO");
}
//输出NO
```

既然有了判断语句那么就一定会有关系运算符 >(大于) <(小于) >=(大于等于) <=(小于等于) ==(等于) !=(不等于) 他和现实中是一样 如果 2 > 1 为真 2 <= 1 为假 以此类推...

运算符产生的结果我们可以使用bool进行存储，或者采用数字型来存储，因为只有真假，在计算机中同样的 1 代表了真，0代表了假

```
bool a = true; //代表真也可以写成 bool a = 1;
bool a = false; //代表假 也可以写成 bool a = 0;
```

下面给出一道题 输入两个整数a，b我们想知道:

1. a是否大于b?
2. a是否小于或等于b?
3. a是否不等于b?

   输出3个整数，用空格隔开，如果成立输出1，不成立输出0;

```
#include<stdio.h>
int main(){
    int a,b;
    scanf("%d %d",&a,&b);
    printf("%d\n",a > b);
    printf("%d\n",a <= b);
    printf("%d\n",a != b);
}
```

那么我们能够判断真假了，我们就可以和我们的如果语句结合起来了
上面的题目也可以这么写

```
#include<stdio.h>
int main(){
    int a,b;
    scanf("%d %d",&a,&b);
    if(a > b) printf("%d\n",1);
    else printf("%d\n",0);
    if(a <= b) printf("%d\n",1);
    else printf("%d\n",0);
    if(a != b) printf("%d\n",1);
    else printf("%d\n",0);
}
```

注意: if语句后面如果不加{}，若为真，他只会执行后边第一句话

```
if(a >= 1) printf("%d",1);
printf("%d",2);
//以上代码如果a < 1的话只会输出2，若a >= 2则会输出 1，2
if(a >= 1);
printf("%d",2);
//以上代码不管a是几都会输出2，因为 if语句把后边的 ；判定为第一句话
```

注意:不能对一个数字同时使用关系运算符 例如

```
如果你想要表达 a > b 并且 b > c不能这么写:
if(a > b > c) //这个运算会先运算 a > b把a > b的结果x 写上再进行 运算 x > c，x的结果只有1，0两种
所以会导致错误。
```

9.1.1 分支嵌套

```
if(a == 1){
    if(a % 2 == 0){
        printf("YES");
    }else{
        printf("NO");
    }
}
```

请问大家什么时候才能实现 a % 2 == 0
当然啥时候抖不能实现，因为要实现a % 2 == 0必须要先让a == 1

```
if(a == 4){
    if(a % 2 == 0){
        printf("YES");
    }else{
        printf("NO");
    }
}
```

请问大家什么时候才能实现 a % 2 == 0
当a = 4的时候才能实现。

9.2 练习

1. 下面的运算是真是假?
    1. 正整数 >= 负数
    2. 整数 >= 负数
    3. 1 >= 2
    4. 2 >= 2
    5. -2 >= -1
    6. -2 < -1
    7. 1 == 2
    8. 1 != 2
    9. 2 > 1 > 1
    10. 10 < 1 < 1
    11. 10 != 2 != 1
    12. 10 != 2 != 0
    13. 2 >= 1 >= 1
    14. 5 > 1 > 2 > 1 > 1 != 0
    15. 2 == 1 == 2 == 3 == 0
2. 下面的这个分支有可能输出什么

    ```
    if(a >= 100){
        printf("我是大聪明");
    }else if(a >= 110){
        printf("我是小聪明");
    }else if(a >= 80){
        printf("小聪明是我");
    }else{
        printf("大聪明是我");
    }
    ```

3. 奇数和偶数区别是什么?
4. 请用代码形式写出： 输入一个整数，若是奇数则输出YES，否则就输出NO

5. 请用代码形式写出 : 输入一个整数，若这个数能够整除 3就输出 aaa，不然如果能整除 2 就输出 cccc，否则输出 dddd

6. 请用代码形式写出： 输入一个整数，若这个数能够整除3并且能够整除2就输出aaa，否则不输出

7. 以下代码合法吗？若合法会产生什么结果？

```
int a = 1;
if(a >= 1) printf("%d",1);
else printf("%d",2);
```

```
int a = 2,b = 1,c = 1;
if(a > b > c)
    printf("aaa");
```

```
int a = 1;
if(a >= 1);
else;printf("%d",1);
```

```
int a = 1;
if(a >= 1); printf("%d",2);
else;printf("%d",1);
```

```
int a = 1,b = 2;
if(a != 1) printf("%d",b);
else printf("%d",a);
```

```
int a = 1,b = 2;
if(a != 1) printf("%d",b);
else if(a != 2) printf("%d",b);
else printf("%d",a);
```

```
int a = 1,b = 2;
if(a != 1){
    printf("YES");
    printf("%d",b);
}else if(b != 2){
    printf("%d",b);
}else{
    printf("NO");
}
```

```
int a = 105;
if(a % 3 == 0){
    if(a % 3 == 0){
        printf("YES\n");
    }else if(a % 5 == 0){
        printf("NO\n");
    }else{
        printf("AAA\n");
    }
```

```
    if(a % 5 == 0){
        printf("YES\n");
    }else if(a % 7 == 0){
        printf("NO\n");
    }else{
        printf("III\n");
    }
    if(a % 5 == 0){
        if(a % 7 == 0){
            printf("YES\n");
        }else{
            printf("NO\n");
        }
    }
}
```

```
if(1 > 2 > 0){
    printf("YES\n");
}else if(2 > 1 > 3 > 0){
    printf("NO\n");
}else if(2 < 1 < 3 != 0){
    printf("AAAA\n");
}else{
    printf("DDDD\n");
}
```

```
if(1 > 2 > 0){
    printf("YES");
}else{
    printf("Asdsa");
}else{
    printf("asddd");
}
```

```
int a = 1;
if(13){
    int a = 2;
    printf("%d",a);
}
```

```
int a = 1;
{
    int a = 3;
    printf("%d",a);
}
```

8. 对于以下代码请分析每一句话是什么意思，如果有运算请写出先运算什么，用什么结果进行下一步运算
例如 1 > 2 > 3 先运算 1 > 2，再运算(1 > 2的结果) > 3

```
if(a != 1 > 2){
    printf("%d",2);
}else if(a != 1 < 2){
    printf("%c",65);
}else{
    printf("DDD!");
}
```

```
if(a * 2){
    printf("YES\n");
    if(a / 2) printf("YES2\n");
    else if(a % 2) printf("YES3\n");
}else if(a % 2 == 0){
    printf("YES3\n");
    if(a % 3 == 0){
        printf("YES4\n");
    }
    if(a % 4 == 0){
        printf("YES5\n");
    }
}
```

9. 请问以下代码是否一定输出YES

```
已知 a % 4 == 0
if(a % 2 == 0){
    printf("YES");
}
```

```
已知 a > 1 <= 2 为真
if(a > 1){
    printf("YES");
}
```

```
已知a > 1 <= 0 为真
if(a > 1);
else{
    printf("YES");
}
```

```
已知 1 > ((2 > a) > 3) 为真
if(a < 2){
    printf("YES");
}
```

10. 请以代码形式写出以下描述（写出最简答案）

```
如果整形a能够整除3并且整形a能够整除5则输出YES
否则如果整形a能够整除3则输出NO
否则输出DDDD
```

```
如果整形a能够整除3并且能够整除5并且能够整除7并且能够整除11并且能够整除13并且能够整除39则输出YES
```

9.2 分支2

9.2.1 与运算和或运算

现有一个问题:

给出两个数字a，b，想要知道:

1. a和b是否都大于0
2. a和b是否有一个大于0

3. a和b是否都不大于0

   如果条件为真则输出1，如果条件为假则输出0

对于这个题目，我们发现一个一个条件不够用了，需要用分支嵌套或者多个条件复合成一个条件进行判断，在汉语的语境下类似 **'且' '或者' '不'**得情况，就需要逻辑运算符进行连接，这样的表达式也称作逻辑表达式。和关系表达式一样，逻辑表达式的结果也是 真(1)和假(0);

```
#include<stdio.h>
int main(){
    int a,b;
    scanf("%d %d",&a,&b);
    printf("%d\n", (a > 0 && b > 0));
    printf("%d\n",(a > 0 || b > 0);
    printf("%d\n",!(a > 0 || b > 0));
    return 0;
}
```

上述代码，我们可以猜测

有以下几个运算符

1. 与运算符 &&（并且)
2. 或运算符 || （或者）
3. 非运算符！ （否）

a	b	a > 0 && b > 0
0	0	0
0	1	0
1	0	0
1	1	1

a.(与运算符)

a	b	a > 0 || b > 0
0	0	0
0	1	1
1	0	1
1	1	1

a.(或运算符)

a	b	!(a > 0 || b > 0)
0	0	1
0	1	0
1	0	0
1	1	0

a.(非运算符)

例如:今天如果不下雨并且我妈妈给我的零花钱超过100块，我就去买小恐龙。这句话里面有两个判断，一个是不下雨，一个是零花钱超过100块，连接起来的符号是并且。那么并且在我们C语言是用 && 来表示的。

```
//用1来表示下雨，0来表示不下雨
if(weather != 1 && money > 100){
    printf("我要去买小恐龙");
}
```

我们发现用并且(&&)连接的两个条件必须都为真，结果才为真

下面给出另一个例子:如果我第一题或者第三题错了，我就是大傻子
这个例子也有两个判断，一个是我第一题错了没有，一个是我第三题错了没有，他们两个是用或者连接的
或者在计算机中是用||来表示的。

```
//我们用1来表示正确，0来表示错误
if(T1 == 0 || T2 == 0){
    printf("我是大傻子");
}
```

我们发现用或者(||)连接的两个条件只要有一个为真，结果就为真

9.2.2 否运算

否运算单独来说，因为和真假有关，之前说过 非0为真，真只能是1，假只能为0。

所以 !0 = 1，!1 = 0，!123 = 0，!-15 = 0。

9.3 练习:

1. (3 >= 2) && (5>= 3) 真还是假?
2. (3 >= 2) && (2 >= 3) 真还是假?
3. (1 >= 2) && (2 >= 3) 真还是假?
4. (3 >= 2) || (5>= 3) 真还是假?
5. (3 >= 2) || (2 >= 3) 真还是假?
6. (1 >= 2) || (2 >= 3) 真还是假?
7. (3 >= 2) && (！(5>= 3))真还是假?
8. (3 >= 2) && (！(2 >= 3)) 真还是假?
9. (1 >= 2) && (！(2 >= 3)) 真还是假?
10. (1 >= 2) || (！(2 >= 3)) 真还是假?
11. (1 > 2 > 3) && (1 < 2 < 3) 是真还是假?
12. 请用代码的形式写出，如果整数a能够同时整除3 5 7，输出Yes，否则输出No

13. 请问以下代码会产生什么结果?

```
int a = 2,b = 1;
if(((a > 1 > 2) != 1) || b > 0){
    printf("YES");
}else{
    printf("NO");
}
```

```
int a = 2,b = 1;
if((a > 1 > (2 != 1)) && b > 0){
    printf("YES");
}else{
    printf("NO");
}
```

```
int a = 2,b = 1;
if(((a > 1 > 2) != 1) || !(b > 0)){
    printf("YES");
}else{
    printf("NO");
}
```

```
int a = 24;
if(a % 2 == 0 && a % 3 == 0 && a % 4 == 0){
    printf("YES");
}
```

14. 请问以下说法怎么写代码?

```
如果整形a的字符形式是A的话并且a是奇数则输出YES
```

```
如果整形a能够整除7或者整形a能够整除3则输出YES
否则如果整形a能够整除7则输出NO
否则输出aaa
```

```
如果整形a能够整除8或者整形a能够整除2则输出YES
否则输出NO
```

10. 循环语句1

10.1 循环的基本使用

如果我们要使得一个变量a + 1 + 2 + ... + 10000，我们有两种办法，一种是自己一个一个加上去，一种是a + (1 + 10000)*10000/2。如果选择第一种方式的话，我们需要写10000个加法会很麻烦。所以我们C语言作者创造了循环，不断地做一件事情。本文重点是for循环语句，一笔带过while循环

循环语句的模板 for(;;){}，循环里面一共有3个合法的语句，分别用两个； 隔开

第一个； 前面写的是在循环开始前的准备活动，相当于一个语句(循环开始前执行，并且只执行一次)，相当于运动会的开幕式

第二个； 前面写循环进行的条件，例如 i <= 4，表示的是变量 i 如果 <= 4的话就继续进行循环，也可以不写(如果不写就相当于一直循环)（每次开始前执行），相当于判断运动会有没有下雨

第二个； 后面写每次循环结束后执行的操作（也可以不写） 相当于运动会扫垃圾，每一次循环结束的时候会执行一次

大括号里面写的就是运动会的项目，重复的做这一件事情

for(int i = 0; i <= 4; i = i + 1) 这句话就表示 我们定义一个整形变量，循环进行的条件是整形变量 i <= 4，每次循环结束后让 i = i + 1

举几个例子

```
for(int i = 0; i <= 4; i = i + 1){
    printf("%d ",i);
}
```
第一次做的事情是先判断i是否小于等于4 然后输出i，此时i = 0，再给i加上1，此时i = 1
第二次做的事情是先判断i是否小于等于4 然后输出i，此时i = 1，再给i加上1，此时i = 2
第三次做的事情是先判断i是否小于等于4 然后输出i，此时i = 2，再给i加上1，此时i = 3
第四次做的事情是先判断i是否小于等于4 然后输出i，此时i = 3，再给i加上1，此时i = 4
第五次做的事情是先判断i是否小于等于4 然后输出i，此时i = 4，再给i加上1，此时i = 5
第六次做的事情是先判断i是否小于等于4 然后发现大于4，退出循环
所以一共循环了5次，输出0 1 2 3 4

```
int sum = 0;
for(int i = 0; i <= 4; i = i + 1){
    sum = sum + 1;
    printf("%d\n",sum);
}
```
第一次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 1，sum = 1
第二次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 2，sum = 2
第三次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 3，sum = 3
第四次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 4，sum = 4
第五次做的事情是先判断i是否小于等于5 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 5，sum = 5
第六次做的事情是先判断i是否小于等于5 然后发现i > 4 所以退出循环

```
int sum = 0;
for(int i = 0; i <= 4; i = i + 1){
    sum = sum + 1;
    printf("%d ",sum);
}
```
第一次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 1，sum = 1
第二次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 2，sum = 2
第三次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 3，sum = 3
第四次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 4，sum = 4
第五次做的事情是先判断i是否小于等于4 发现大于4退出
最后输出 1 2 3 4 5
我们发现写在循环外边的变量不会被重新定义，还会保留原来的值

```
int sum = 0;
for(int i = 0; i <= 2; i = i + 1){
    int x = 1;
    x = x + 1;
    printf("%d ",x);
}
```
第一次做的事情先判断i是否小于等于2 然后定义一个整形变量x，并且赋值为1 然后给x加上1，输出x，最后给i加上1
第二次做的事情先判断i是否小于等于2 然后定义一个整形变量x，并且赋值为1 然后给x加上1，输出x，最后给i加上1
第三次做的事情先判断i是否小于等于2 然后定义一个整形变量x，并且赋值为1 然后给x加上1，输出x，最后给i加上1
第四次做的事情先判断i是否小于等于2 然后发现i > 2所以退出循环
最后输出 2 2 2
我们发现写在循环里面的变量，每次会重新定义，所以进到下一次循环后，上一次的变量就不存在了。

以上的循环语句我们明确知道会循环几次，这叫做次数循环语句

还有一种循环语句叫条件循环语句 while(){},while循环和for循环不同的是，只需要写循环进行下去的条件（为真继续执行），每次循环前会判断一次

while()，while括号里面的就是条件执行语句，例如 while(a >= 1) 的意思就是 如果 a >= 1的话循环就继续执行

所以while(i <= 4) = for(;i <= 4 ;)

举个例子

```
int i = 0;
while(i <= 4){
    printf("%d ",i);
    i = i + 1;
}
//以上代码答案会输出 0 1 2 3 4，因为i = i + 1写在输出后边
int i = 0;
while(i <= 4){
    i = i + 1;
    printf("%d ",i);
}
//以上代码答案会输出 1 2 3 4 5，因为i = i + 1写在输出前边
```

那么我们想要把整形数组的 第一个~第五个位置都填上我们就要这样写

```
for(int i = 0; i < 5; i = i + 1){

•       scanf("%d",&a[i]);

}

int i = 0;
while(i < 5){
    scanf("%d",&a[i]);
}
```

注意：如果不写大括号的话，循环语句只会执行 后边的第一句话

```
for(int i = 1; i <= 10; i = i + 1）printf("%d\n",1);
printf("%d\n",2);
//这样子会输出10个1，1个2
for(int i = 1; i <= 10; i = i + 1) ;
printf("%d\n",1);
//这样子只会输出一个1，因为我们计算机是通过判断有没有；来看是否是一句话
```

10.2 循环里外的变量

```
int i = 1;
for(; i <= 2; i = i + 1){
    int j = 1;
    j = j + 1;
    printf("%d %d\n",j,i);
}
//最后输出
//2 1
//2 2
//我们发现定义在循环里面的变量值并没有变大
//说明每次循环会重新定义循环里面的变量，所以循环里面的变量作用域和生命周期是声明开始，此次循环结束
```

10.3 疑惑点

1.既然循环里面是和外面一样的语句，那么能否在循环里面写输出语句?

```
int i = 1;
for(printf("1");printf("aa"),i <= 2;printf("dd"),i++)
//我们发现最后输出了1aaddaadd
//只要合法就都可以写，但是注意每一块语句的运行次数和时机
```

2.int i写在循环第一句和写在外边有什么区别?

```
int i = 1;
for(;i <= 2; i++);
printf("%d",i);

for(int i = 1; i <= 2; i++);
printf("%d",i);
//我们发现这两个代码，一个可以输出3，一个编译不成功
//说明定义在里外作用域和生命周期不同，定义在第一句里面，只能在循环里面使用，并且只声明一次，生命周期是整个循环。
```

10.4 练习:

1. 请用代码形式写出: 输入字符填上字符数组a的1~10个位置

2. 请用代码形式写出: 输入10个整数若是奇数则输出YES否则输出NO，每输出一行换一行

3. 请思考如何在C语言中判断是否为质数，请用代码形式写出

4. 给你1000个整数，从这1000个数中找到其中的最小值，最大值并且输出出来

5. 《庄子》中说到，“一尺之棰，日取其半，万世不竭”。第一天有一根长度为 a 的木棍，从第二天开始，每天都要将这根木棍锯掉一半（每次除 2，向下取整）。第几天的时候木棍的长度会变为 1？

6. 以下代码循环几次？转换成while循环怎么写或者转换成for循环怎么写；

```
for(int i = 1; i <= 10; i = i + 1);
```

```
for(int i = 0; i < 10; i = i + 1);
```

```
for(int i = 1; i <= 9; i = i + 2);
```

```
for(int i = 1; i <= 9; i = i * 2);
```

```
for(int i = 1; i < 9;i = i + 1, i = i * 3);
```

```
int i = 1;
while(i <= 5){
    i = i + 1;
    i = i * 2;
}
```

7. 以下操作是否合法？若合法输出的结果又是什么？转换成while循环怎么写？

```
for(int i = 1,j = 1; i <= 10, j <= 11; i = i + 1, j = j + 1){
    printf("%d\n",j);
}
```

```
for(int i = 1,j = 1; i <= 10 && j <= 11; i = i + 1, j = j + 1){
    printf("%d\n",j);
}
```

```
for(int i = 1,j = 1; i <= 10 || j <= 11; i = i + 1, j = j + 1){
    printf("%d\n",j);
}
```

```
int i = 1;
for(printf("Hello"); i <= 5,printf("LL"); i = i + 1);
```

```
int i = 1;
for(printf("Hello"); i <= 5; i = i + 1,printf("LLL"));
```

```
for(int i = 1,printf("LLLL"); i <= 2; i = i + 2);
```

```
int i = 1,j = 1,h = 1;
for(; i <= 100,j <= 100,h <= 2;){
    h = h * 2;
    printf("Hello");
}
```

```
for(int i = 1; i <= 5; i = i + 1);
printf("%d",i);
```

```
for(int i = 1; i <= 10; i = i + 1){
    i = i + 1;
    printf("%d\n",i);
}
```

8. 以下操作是否合法？若合法输出的结果又是什么？转换成for循环怎么写？

```
int i = 1;
while(i <= 5){
    printf("%d",i);
    i = i + 1;
}
```

```
int i = 1;
while(i <= 5)
    printf("%d",i);
    i = i + 1;
```

```
int i = 1,j = 1;
while(i <= 5,j <= 10){
    printf("%d",i);
    i = i + 1;
    j = j + 1;
}
```

```
int i = 1;
while(i <= 5 && j <= 10){
    printf("%d",i);
    i = i + 1;
    j = j + 1;
}
```

```
int i = 1;
while(i <= 5 || j <= 10){
    printf("%d",i);
    i = i + 1;
    j = j + 1;
}
```

```
int i = 1;
while(i <= 5,i++){
    printf("%d\n",i);
}
```

```
int i = 1;
while(i++,i <= 6){
    printf("%d\n",i);
}
```

循环语句2

循环能和很多东西搭配在一起使用，最主要的就是循环可以和输入，输出， if语句，循环语句本身，数组等一些其他语句搭配在一起使用。

循环 与 输入输出

我们只输入一两个整数的时候，我们可以直接用scanf这样很方便，但是当我们要输入成千上百，或者输出成千上百个数的时候我们单单的去用手是不合适的，这个时候就要用到我们的循环，如下题目：

输入1000个整数，并且将他们全都按照输入顺序输出出来。

```
//对于这个问题我们首先要思考如何输入1个数，输出一个数，然后再思考如何输入1000个数，输出一个数
int x;
scanf("%d",&x);
printf("%d,x);
//如上就输入输出了一个数，那么我们要做这件事情做1000次，只要满足我的循环能够1000次即可
for(int i = 1; i <= 1000; i = i + 1){
    int x;
    scanf("%d",&x);
    printf("%d,x);
}
```

循环 与 判断语句

同样的我们直接通过题目来看这个例子：

输入1000个整数，判断他们每个数，如果是奇数则输出YES，否则输出NO。

```
//同样的我们先思考输入 一个数，如何判断这一个数是奇数还是偶数
int x;
scanf("%d",&x);
if(x % 2 == 1){
    printf("YES");
}else printf("NO");
//如上我们就解决了如何去判断一个数，那么我们只要将这件事情重复做1000次是不是就可以了？
for(int i = 1; i <= 1000; i = i + 1){
    int x;
    scanf("%d",&x);
    if(x % 2 == 1){
        printf("YES");
    }else printf("NO");
}
```

循环 与 数组

循环与数组就有意思了，虽然我们的循环每一次做的事情是一样的，但是经过一次循环后，我们不能保证变量不会改变，我们可以通过这个变量改变机制使用我们的数组，我们同样通过题目来看：

输入1000个整数，反着输出他们每个数

```
//首先我们要反着输出这1000个数字，我们就要把他们都记录下来，才能反着输出，那么记录就是通过数组记录
//我们先考虑输入一个数字我们可以存在哪
//假设我想要输入的第 i 个数存在 数组第 i 个位置上
int a[1010];
scanf("%d %d %d",a[1],a[2],a[3]);
//那么如上我分别把第一个数存在了第一个位置，第二个数存在了第二个位置，第三个数存在了第三个位置，有一个规律就是数组中的下标每输入一个数下标增加1
//利用这个规律，我们发现循环可以做到让 变量每次增加相同的数或者不同的数
//假如我让 i 来代表数组的下标，那么我只要让输入第一个数的时候 i 为 1，第二个数的时候 i 为 2即可
for(int i = 1; i <= 1000; i = i + 1){
    scanf("%d",&a[i]);
}
```

11. 变量的作用域和生命周期

这个产生的区别最主要就发生在我们的{ }和一些循环，分支语句中。
首先{ }是包含的意思，包含在内部的意思，循环分支如果不写大括号，默认包含后面的第一句话

11.1 作用域

作用域，程序设计概念，通常来说，一段代码中所用到的名字并不总是有效/可用的
而限定这个名字的可用性的代码范围就是这个名字的作用域
我们可以试一下以下代码

```
#include <stdio.h>

int main(){
    {
        int num = 0;
    }
    printf("%d\n",num);
}
```

我可以发现爆出了错误，如果我们改成以下代码我们就发现代码能够运行。

```
#include <stdio.h>

int main(){
    {
        int num = 0;
        printf("%d\n",num);
    }
}
```

所以我们可以把作用域抽象成这些名字可以在哪个地方使用。
大括号内部的只能在这个大括号内部里面使用

我们称写在大括号里买的变量为局部变量
写在大括号外边的就是全局变量
局部变量的作用域是变量所在的局部范围，全局变量的作用域就是整个工程
同样的，因为我们的循环，分支语句全是有大括号的，所以在这里面定义的变量，只能在这里面使用

11.2 生命周期

变量的生命周期指的是变量的创建到变量的销毁之间的一个时间段

1. 局部变量的生命周期是：进入作用域生命周期开始，出作用域生命周期结束。
   通过作用域的学习，我们发现不管是循环语句，判断语句里面写的变量只能在他们内部使用。
   所以我们出了他们内部，我们再也不会用他们的这些变量，所以系统会判断他为垃圾，自动销毁回收了
2. 全局变量的生命周期是：整个程序的生命周期。只有总程序结束了才会销毁

11.3 练习:

1. 以下哪些操作是合法的，并且说出每个变量的生命周期和作用域

```
#include<stdio.h>
int b = 1;
int main(){
    int a = 1;
    {
        printf("%d",b);
        printf("%d",a);
    }
}
```

```
#include<stdio.h>
int main(){
    {
        int a = 1;
        printf("%d",a);
    }
    printf("%d",1);
}
```

```
#include<stdio.h>
int main(){
    {
        int a = 1;
    }
    printf("%d",a);
}
```

```
#include<stdio.h>
int main(){
    if(1){
        int a = 1;
    }
    printf("%d",a);
}
```

```
#include<stdio.h>
int main(){
    for(int i = 1; i <= 10; i = i + 1){
        int a = 1;
    }
    printf("%d",i);
    printf("%d",a);
}
```

```
#include<stdio.h>
int main(){
    {
        int a = 1;
    }
    for(int i = 1; i <= 10; i = i + 1){
        a = a + 1;
    }
    printf("%d",a);
}
```

```
#include<stdio.h>
int main(){
    for(int i = 1; i <= 10; i = i + 1){
        int a = 1;
        {
            a = a + 1;
        }
        printf("%d",a);
    }
}
```

12. 循环中的关键词

有两个我们常用的关键词一个是continue，一个是break

1. continue是跳过的意思，也就是说跳过此次循环的意思。举个例子

```
for(int i = 1; i <= n; i = i + 1){
    sum += 1;
    continue;
    sum += 2;
}
//这个代码，sum += 2不会执行，因为每次遇到了continue就跳过了此次循环
```

2. break是结束的意思，也就是说结束循环。举个例子

```
for(int i = 1; i <= n; i = i + 1){
    sum += 1;
    break;
    sum += 2;
}
//这个代码，sum += 1只会执行一次，因为遇到了break，所有循环结束
```

12.1 练习：

1. 以下代码会输出什么?

```
for(int i = 1; i <= 5; i = i + 1){
    sum += 1;
    break;
    sum += 2;
}
printf("%d",sum);
```

2. 以下代码会输出什么?

```
for(int i = 1; i <= 5; i = i + 1){
    sum += 1;
    continue;
    sum += 2;
}
printf("%d",sum);
```

3. 输入一个整数，如果这个整数不是1，继续输入，重复这个动作。

4. 以下代码循环几次?

```
for(int i = 1; i <= 2; i = i + 1){
    if(i == 1) break;
}
```

```
for(int i = 1; i <= 9; i = i + 2){
    if(i % 2 == 0) break;
}
```

```
for(int i = 1; i <= 10; i = i + 1){
    if(i % 2 == 1) continue;
}
```

13. 分支3

在分支2的时候我们学了 &&(与运算符)，||(或运算符)。那么这堂课我们要讲一下基本逻辑。

首先 &&和||的优先级是一样的，倘若没有括号我们是从前往后运算。

请大家回答以下 运算结果答案是否相同?

```
(a >= 0 && b >= 0) && c >= 0
a >= 0 && b >= 0 && c >= 0
```

```
a >= 0 && (b >= 0 && c >= 0)
a >= 0 && b >= 0 && c >= 0
```

```
a >= 0 || b >= 0 || c >= 0
(a >= 0 || b >= 0) || c >= 0
```

```
a >= 0 || b >= 0 || c >= 0
a >= 0 || (b >= 0 || c >= 0)
```

```
a >= 0 && (b >= 0 || c >= 0)
a >= 0 && b >= 0 || c >= 0
```

```
(a >= 0 && b >= 0) || c >= 0
a >= 0 && b >= 0 || c >= 0
```

```
a >= 0 && b >= 0 || c >= 0
c >= 0 || a >= 0 && b >= 0
```

13.1 练习:

1. 当a = 3,b = 4,c = 5判断以下表达式是否成立？ (乘除运算优先级最高，加减其次，最后是关系运算符)
   (1) a < b || a > c || a > b
   (2) a > c || b > a && c > b
   (3) b - a == c - b
   (4) a * b - c > a * c - b || a * b + b * c == b * b * (c - a)
2. 当 a = 1,b = 0,c = 1，判断以下表达式是否成立?
   (1) !a || ! b (先运算 !)
   (2) (a && !a) || (b || !b)
   (3) a && b && c || !a || !c
   (4) a && (b && c || a && c)
   (5) !b && (c && (a && (!c || (!b || (!a)))))

14. 运算符的使用

之前讲过 + - * / %，他们都是有两个操作数的，什么叫两个操作数呢?
例如 a + b，a是一个数字，b也是一个数字，所以叫做有两个操作数。
那么有两个操作数的被称作二元操作符。今天着重要讲的是，一元操作符和三元操作符。
顾名思义，就是一个操作数的和三个操作数的，因为现实数学中用不到，**所以以下知识请不要在数学课上使用!**

14.1 一元操作符

一元操作符有很多，下面介绍几个常用的

1. ++（称作自增，通过自增可以使变量在自身的基础上加1）

   使用方法为， ++a，a++，那么++a和a++有什么区别呢?
   咋们可以通过两段代码来看出区别

   ```
   int a = 1;
   int b = ++a;
   printf("%d\n",b);
   b = a++;
   printf("%d\n",b);
   //我们发现这两段代码的输出都是2，这下就可以看出来，++a在使用前立刻会增加1然后再使用，a++是在使用后立刻会增加1。
   ```

2. --（称作自减，通过自减可以使变量在自身的基础上减1）

   使用方法为，--a，a--，那么--a和a--有什么区别呢?
   咋们可以通过两段代码来看出区别

   ```
   int a = 1;
   int b = --a;
   printf("%d\n",b);
   b = a--;
   printf("%d\n",b);
   //我们发现这两段代码的输出都是0，这下就可以看出来，--a在使用前立刻会减少1然后再使用，a++是在使用后立刻会减少1。
   ```

3. ！（称作否，因为计算中有两种状态，一种为真一种为假，所以可以用它来表示相反的状态）

那么对于!号的使用，我们首先知道 一开始我们教大家， 1代表真，0代表假所以 ！1 = 0， ！0 = 1这个没有问题
那么对于 ！x 如果这个x不是1，也不是0怎么办？
所以为了不产生错误，c语言干脆把 不等于0的数字都当作真，0是假
所以 ！0 = 1，！1 = 0，！2 = 0，！-1 = 0

4. ~ （称作取反，它其实是对于二进制进行取反。）

举个例子 4的二进制补码是 00000000000000000000000000000100，因为int类型是32位，所以我们存进去是这么多位
那么 ~4 = 11111111111111111111111111111011 = -5，怎么快速的看出答案呢，首先是补码运算，所以全为1的时候 = -1，-1再减去4不就是-5嘛？

14.2 三元操作符

三元运算符可有意思了很简单，他包含了一个条件表达式和两个结果。
三元运算符是软件编程中的一个固定格式，语法是“条件表达式?表达式1:表达式2”。使用这个算法可以使调用数据时逐级筛选。
表达式：“()? :”。
()中进行二元运算
?再运算,就形成三元运算符
举个例子

```
int b = (2 > 1)? 1 : 2;
printf("%d\n",b);
//这段代码会输出2，因为三元运算符只有为真才会执行？后这个结果，否则执行 ： 后的结果
int b = (1 > 2)? 1 : 2;
printf("%d\n",b);
//这段代码就会输出1
```

14.3 二元运算符

以下是一写比较常用得二元运算符

1. &（按位与运算，它也是对于二进制来说的，对于二进制的每一位进行 &&运算）

&&运算是有0则为0
例如 1111000 & 0101100 = 0101000

2. | （按位或运算，它也是对于二进制来说的，对于二进制的每一位进行 || 运算）

||运算是有1则为1
例如 1111000 | 0101100 = 1111100

3. ^（按位异或运算，它也是对于二进制来说的，对于二进制的每一位进行 ^ 运算）

异或运算比较特殊，他是如果两位不相同则为1，两位相同则为0
举例子 0 ^ 0 = 0,0 ^ 1 = 1,1 ^ 1 = 0
那么 1111000 ^ 0101100 = 1010100

4. <<（称作左移，它也是对于二进制来说的，把二进制整体往左边移一位，右边填0）

举个例子 4的二进制补码是 0100，那么 0100<<1 = 1000 = 8就是整体左移一位，整体乘2，但是小心溢出

5. >>（称作右移，它也是对于二进制来说的，把二进制整体往右边移一位，左边补充和首位一样的数字）

举个例子，-4的二进制补码可以是 1100，1100>>1 = 1110 = -2，我们把最左边的那一位补上了1，如果是0的话答案就等于6了
4的二进制补码是0100，0100>>1 = 0010 = 2，我们把最左边的那一位补上了0，如果是1的话那就等于-2了。整体整除2.
大家还记得，我们的C语言中的 / 是整除的意思，向0取整，这就可以通过右移来看，因为计算机的运算其实最后都会转换成左移右移
例如3的二进制补码可以是 011，011>>1 = 001，最右边那一位是1，但是右移后没有他的位置了，所以直接没了，所以最后答案等于1，相当于向下取整
例如-3的二进制补码可以是101，101>>1 = 110，最右边那一位是1，但是右移后也没有他的位置了，所以直接没了，所以最后答案等于-2，相当于向下取整
此时有同学就有疑问了，负数整除不是向上取整么。
对，因为我们现实中为了保证 被除数 = 除数 * 商 + 余数，所以我们才保证向上取整

6. ，（称作逗号运算符）

逗号的作用比较特殊
1.它保证了被他分割的表达式从左往右取值,也就是逗号左侧的表达式都是在左侧完成
2.逗号表达式的值是右侧项的值;
例如下面这个语句

```
x = (y = 3,(z = ++y + 2) + 5);
```

的效果是，先把3赋值给y，递增y为4，再把4 + 2只和(6)赋值给z，接着加上5，最后把结果11赋值给x。
另一方面假设在写数字的时候不小心写了逗号

```
a = 123,456;
```

那么C编译器会把它当作一个逗号表达式，即a = 123是左侧的表达式，456是右边表达式
因此与下面的代码效果相同

```
a = 123;
456;
```

3.逗号当作分隔符，例如printf，scanf

其实大家会的差不多了，不过还有更方便的写法现在交给大家比如：大家一般这么写

```
int a = 10;
int b = 2;
b = b + a;
b = b - a;
b = b ^ a;
b = b | a;
b = b & a;
b = b / a;
b = b % a;
//但是我们也可以如下这么写
b += a;//这句话同样表示给变量b加上a，
b -= a;
b ^= a;
b |= a;
b &= a;
b /= a;
b %= a;
```

优先级	运算符	结合律
1	后缀运算符：[] () . -> ++ --(类型名称){列表}	从左到右
2	一元运算符：++ -- ! ~ + - * & sizeof_Alignof	从右到左
3	类型转换运算符：(类型名称)	从右到左
4	乘除法运算符：* / %	从左到右
5	加减法运算符：+ -	从左到右
6	移位运算符：<< >>	从左到右
7	关系运算符：<<= >>=	从左到右
8	相等运算符：== !=	从左到右
9	位运算符 AND：&	从左到右
10	位运算符 XOR：^	从左到右
11	位运算符 OR：|	从左到右

14.4 练习：

1. 思考以下代码会输出什么

```
for(int i = 1; i <= 5; i++) printf("%d\n",i);
```

2. 思考以下代码会输出什么

```
for(int i = 1; i <= 5; ++i) printf("%d\n",i);
```

3. ！10000000，！-100000000，！9999999999999 分别输出什么?

4. 整形类型下~1， ~15 ， ~-5分别输出什么

5. ~(1000111)补， ~(0111111)补， ~(11111111)补分别输出什么

6. 123456 第1位，第2位，第3位，第4位，第5位，第6位如何取？用代码形式写出来

7. (101101)补 & (1110111)补，15 & 14 ，20 | 15，0 | 100，0 & 100，0 ^ 100，100^100， 100 ^ 100 ^ 100，100 & 0 | 100 分别是多少

8. (2 != 1) ? 1 : 2 ，(5 > 2) ? 3 : 2 ，(1 + 2 != 3 > 4) ？ 1： 2。答案分别是什么

9. 10 >> 1， 15 >> 1 ， 10 << 2， 100 << 2，-100 / 2， -10 % 2，10 % 2 分别输出什么(/是C语言中的整除)

10. 对于整形变量X，Y，写出与判断以下性质对应的表达式
    (1) x是否为偶数
    (2) x是否为4位整数
    (3) 得到X在二进制形式下的 第3位
    (4) 判断X在二进制形式下的 第4位是否为1
    (4) 得到X，Y在二进制下的第 3 位的异或值
11. 编写代码实现：求一个整数存储在内存中的二进制（补码）中1的个数

12. 不能创建临时变量（第三个变量），实现两个数的交换。

13. 输入一个整数，把他转换成二进制

14. 一串二进制如何转换成十进制

15. 字符数组，内存，以及数组的构造

为什么要把字符数组单独拿出来说呢，因为在外国人得眼里中文其实是不好表示的，我们可以看我们之前的ascii码中并没有映射中文，所以美国人用几个ascii码来表示一个中文，例如他用32位来表示一个中文，但是一个char是8位，所以需要4个char才可以表示一个，但是我们分开来存的时候，如何把中文映射出来呢？这个时候就需要我们的字符串输入和字符串输出登场了

```
char a[10]; scanf("%s",&a);printf("%s",a);
```

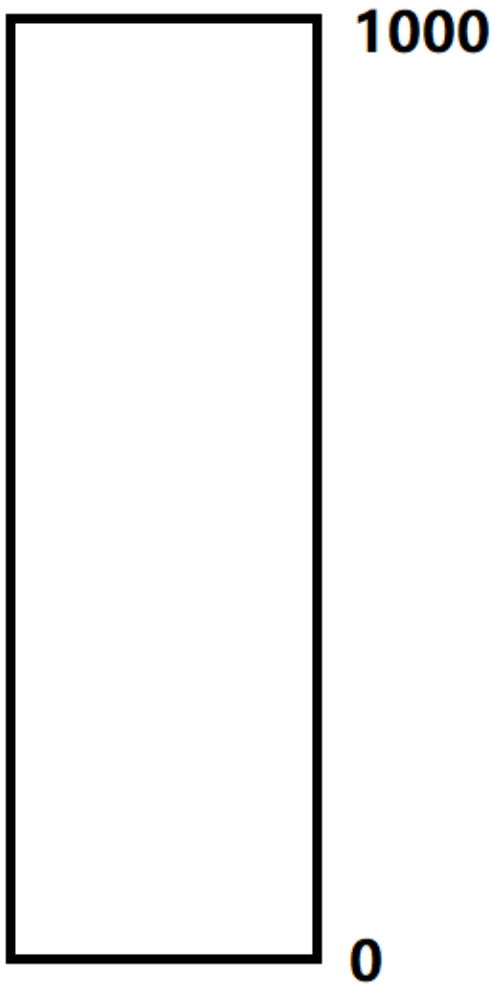
由于字符只能存一个，所以想存多个字符的话，可以使用字符数组。那么输入的字符就一个个分别存在a[0],a[1]....

同学们可能有一些疑问：

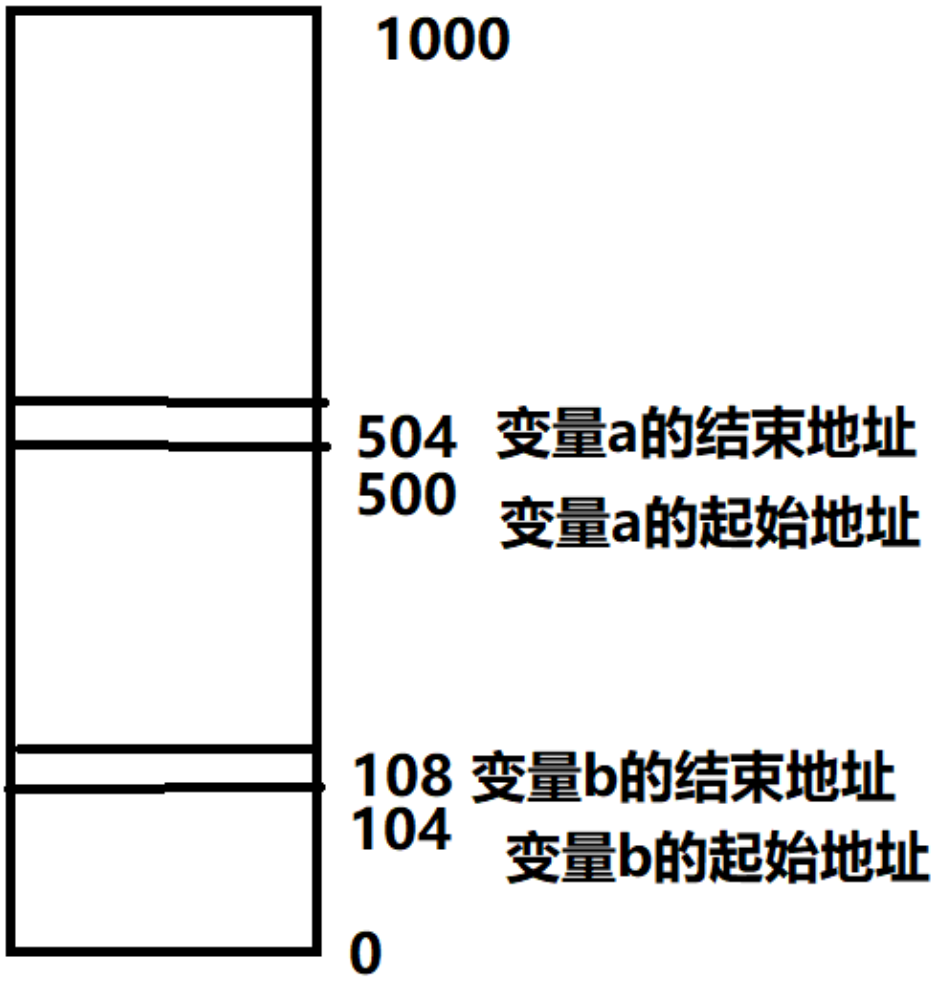
- **为什么字符数组输出名字就能输出所有值?**
- **字符数组怎么判定输出结束的?**

15.1 内存，地址的概念

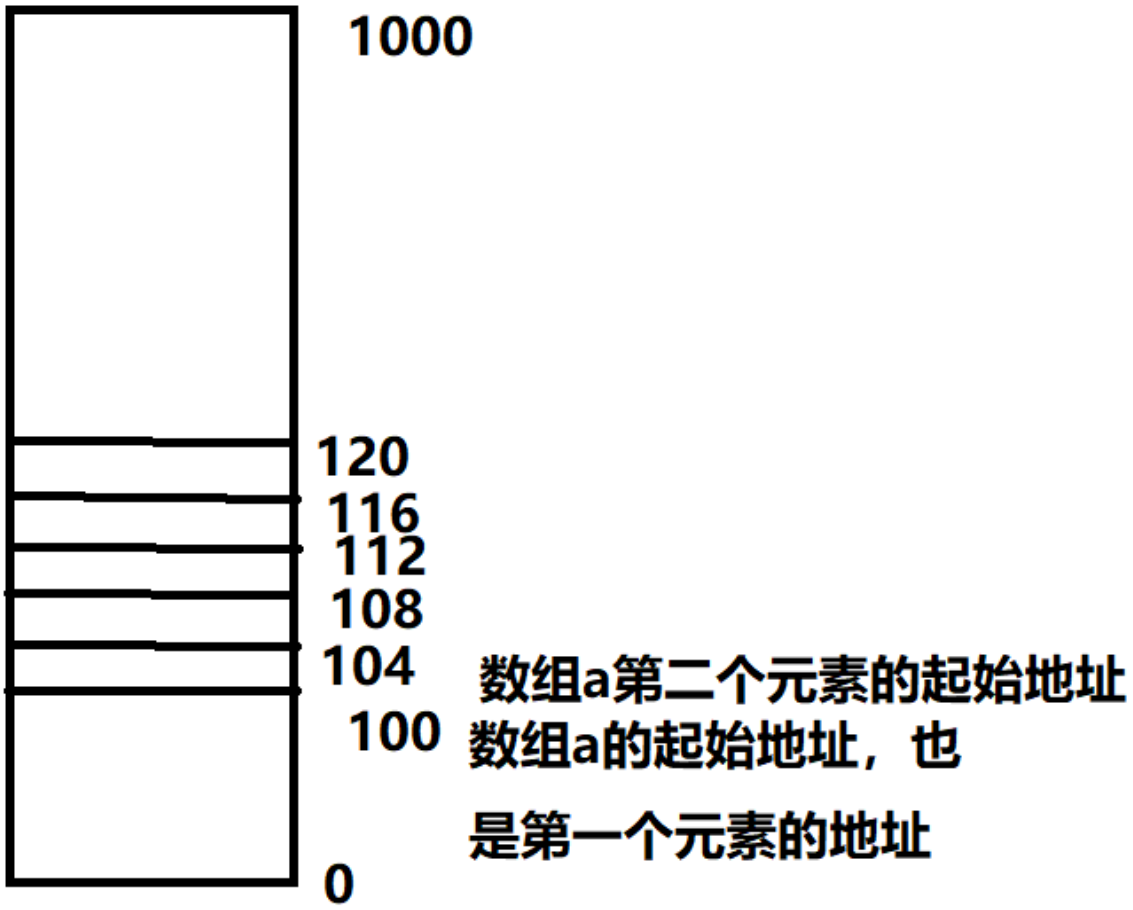
每个计算机的内存是有限的，所以每个变量，每个数组，每句代码都会占用一定的空间，并且都会有地址，那么空间和地址到底是怎么分配的呢？
那么，我们假设我们的空间和我们的商品房一样



假设我们定义了一个整形变量a，一个整形变量b，那么他们两个会在这个空间里面找到一块地方放下。



那么数组的空间是怎么存储的呢？众所周知，数组是很多相同的变量组成的，所以他们的空间是连续的。
例如我定义了 int a[5];



15.2 为什么字符数组可以使用名字输出所有值，而整形数组不行?

首先变量的名字，代表了它的内容，但是数组包含了太多内容，例如整形数组，包含了好多整数，如果把他们直接输出的话，连在一起就产生了歧义，所以C语言作者让数组的名字去代表数组的地址。
然后因为C语言中没有专门处理字符串的东西，所以就选择了字符数组去处理，因为字符串是一整个可以连在一起的，拼接在一起的，但是像整数，浮点数，如果拼接在一起的话他们的意思就会改变，并且再分开也不知道如何分开了。所以C语言干脆就把字符数组单独做一个处理，输出字符串。

大家思考一个问题 : 如果我定义了一个变量a,他的首地址是0，他的末尾地址是4，那么我们知道 1 2 3 有任何意义吗?
答案是没有任何意义，因为我们无法通过 1 2 3推导出任何信息。所以C语言干脆就把 地址中的 + 1 - 1的意思给改变了
他把意思变成了: 以当前数据大小为单位进行加减

```
int a[2];
printf("%d %d %d %d\n",&a,&a + 1,&a[0],&a[0] + 1); //大家觉得有什么区别呢？ 试验一下
```

最后我们发现，&名字 代表的是当前整个元素的地址(元素是整个物体)

15.3 内存进阶内容

```
int a[2];
printf("%d %d %d %d",a,&a,a + 1,&a + 1); //大家觉得他们之间有什么关联?
```

提示1：首先大家思考，a是什么，a[0]又是什么?

最后我们发现a也代表了数组第一个元素的地址，而不是数组的首地址(虽然他们的的值相同，但是意思不同)

15.4 总结:

1. 定义一个数组a，&a代表的是整个数组，而不是数组的元素
2. 定义一个数组a，a也可以代表数组第一个元素的首地址
3. 地址中加减的单位变成了，以我当前元素的数据大小为基础单位
4. 字符数组可以用名字直接输出字符串，其他不行

15.5 字符数组如何判定结束

字符数组是用 '\0'来判定结束的

```
char a[10];
假设a[0] = 'A',a[1] = 'B',a[2] = 'C',a[3] = 'd',a[4] = '\0',a[5] = 'a';
printf("%s",a)；他会输出 ABCd
```

```
那么输入的时候
scanf("%s",a);
他会自动把输入的最后一个字符的后一个位置改成 '\0'，所以我们输出的时候自动会停止
```

同时从这一块我们也能发现，如果给整形数组，浮点型数组也按照顺序输出的话，怎么样停止呢？是用0表示停止还是-1表示停止呢？一看就不合理

15.6 数组的构造

首先**字符数组的构造**我们已经会了，当我的字符数组等于一个字符串的时候，他自动会把字符串拆分字符分别存进去

```
char a[10] = "123";
printf("%c %c %c",a[0],a[1],a[2]); //输出 1 2 3
```

整形数组和浮点数数组因为我们数字需要隔开，所以我们可以使用大括号包含我们想要构造的数字，然后中间用逗号隔开
例如

```
int a[10] = {0,1,2}; //默认从a[0]开始给，没有给到的赋值为0
```

浮点数数组也是一样

```
float a[10] = {0,1,2}; //默认从a[0]开始给，没有给到的赋值为0
```

15.7 练习:

1. 如何在定义的时候把大小为100的整形数组里的元素全都变成0

2. float a[10]；printf("%d %d %d %d %d %d",a,a + 1,&a,&a + 1,&a[0],&a[0] + 1)；(假设数组a的首地址为0，请写出预期输出值)

3. int a = 2; printf("%d %d %d",a,&a,&a + 2); (假设a的首地址为0，请写出预期输出值)

4. int a[10] = {1,2,3}; printf("%d %d %d",a[0],a + 8,&a[8]); (假设a的首地址为0，请写出预期输出值)

5. 以下操作合法吗？若合法输出什么答案

```
scanf("%s",&a); //输入1234
printf("%s",a);
```

```
scanf("%s",a); //输入1234
printf("%s",a);
```

```
scanf("%s",a); //输入1234
printf("%s",a);
```

```
scanf("%s",&a); //输入1234
printf("%s",&a);
```

```
scanf("%s",a); //输入1234
printf("%s",&a);
```

```
scanf("%s",a); //输入1234
printf("%s",&a + 1);
```

```
char a[10] = {1,2,3,4,5,'\0'};
printf("%s",a + 1);
```

```
char a[10] = {1,2,3,4,5,'\0'};
scanf("%s",a + 1); //输入12345
printf("%s",a + 1);
```

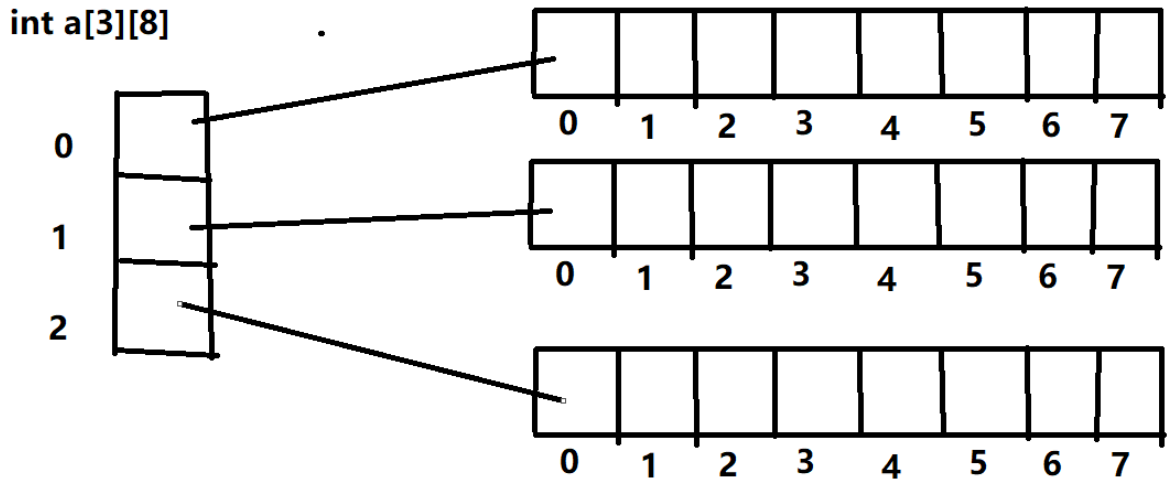
```
char a[10] = {1,2,3,4,5,'\0'};
scanf("%s",a + 1); //输入1234
printf("%s",a + 1);
```

```
char a[10] = {1,2,3,4,5,'\0'};
a[2] = '\0';
scanf("%s",a + 1); //输入12345
printf("%s",a + 1);
```

```
int a[10] = {0,1,2,3};
scanf("%d",a + 2);
printf("%d",a[0]);
```

16. 数组嵌套，循环2

数组嵌套顾名思义: 数组里面套一个数组



读作: 大小为3的数组a里面的每个元素是大小为8的整形数组，也可以读作3行8列的二维数组a

所以我们数组a里面的元素不是整形而是整形数组，数组里面的数组的元素才是整形

就好比学校的元素是班级，班级里面的元素才是人

那么我们想要使用我们的二维数组其实是一样 a[ ][ ]

我们想要输入a数组里第一个元素里的第一个元素 scanf("%d",&a[0] [0]);

如果数组大小是 int a[100] [100];单重循环来输入也是件麻烦事，所以我们会使用双重循环

双重循环:顾名思义 循环里面套循环

每次循环先把里面的循环结束后，在继续当前循环

我们想要给数组int a[100] [100]全部填上我们有两种写法

1. 先把每一行给填上

```
for(int i = 0; i <= 99; i++){
    for(int j = 0; j <= 99; j++){
        scanf("%d",&a[i][j]);
    }
}
```

2. 先把每一列给填上

```
for(int i = 0; i <= 99; i++){
    for(int j = 0; j <= 99; j++){
        scanf("%d",&a[j][i]);
    }
}
```

16.1 练习

1. 请用代码形式输出100以内的质数 (通过判断是否为质数联想)

2. 请用代码形式输入大小为100 * 100 * 100 得三维数组 (输入满)

3. int a[10] [20]里，a[0]~a[9] 装的元素是什么呢？如何表示

4. 若a的首地址为0，假设a里面的每个元素都是为0 下面代码预期值应该是多少？

```
int a[2] [2];
printf("%d %d %d %d %d %d %d %d %d %d",
a,a + 1,&a[0],&a[0] + 1,a[0],a[0] + 1,a[0] [0],a[0] [0] + 1,&a[0] [0],&a[0] [0] +
1);
```

```
int a[4] [4];
printf("%d %d %d %d %d %d %d %d %d %d",
a,a + 1,&a[2],&a[2] + 1,a[2],a[2] + 1,a[1] [0],a[2] [1] + 1,&a[1] [2],&a[3] [1] +
1);
```

5. 请同学画一下，int a[2] [3] [4]。数组长什么样。

17. 时间复杂度和空间复杂度的概念

17.1 时间复杂度

在计算机科学中，**时间复杂性**，又称**时间复杂度**，算法的**时间复杂度**是一个函数，它定性描述该算法的运行时间。这是一个代表算法输入值的字符串的长度的函数。时间复杂度常用大O符号表述，不包括这个函数的低阶项和首项系数。使用这种方式时，时间复杂度可被称为是渐近的，亦即考察输入值大小趋近无穷时的情况。

17.1.1 常数时间

常数时间

播报 编辑

若对于一个算法，$T(n)$的上界与输入大小无关，则称其具有**常数时间**，记作$O(1)$时间。一个例子是访问数组中的单个元素，因为访问它只需要一条指令。但是，找到无序数组中的最小元素则不是，因为这需要遍历所有元素来找出最小值。这是一项线性时间的操作，或称$O(n)$时间。但如果预先知道元素的数量并假设数量保持不变，则该操作也可被称为具有常数时间。

虽然被称为"常数时间"，运行时间本身并不必须与问题规模无关，但它的**上界**必须是与问题规模无关的确定值。举例，"如果a > b则交换a、b的值"这项操作，尽管具体时间会取决于条件"a > b"是否满足，但它依然是常数时间，因为存在一个常量t使得所需时间总**不超过**t。

举个例子

```
for(int i = 1; i <= n; i++) 就是o(n)
printf("%d",1); 就是 o(1)
```

17.1.2 对数时间

对数时间

播报 编辑

若算法的$T(n)$ =O((logn)^k)，则称其具有**对数时间**。由于计算机使用二进制的记数系统，对数常常以2为底（即$\log_2 n$，有时写作lg$n$）。然而，由对数的换底公式，$\log_a n$和$\log_b n$只有一个常数因子不同，这个因子在大O记法中被丢弃。因此记作O((logn)^k)，而不论对数的底是多少，是对数时间算法的标准记法。

常见的具有对数时间的算法有二叉树的相关操作和二分搜索。

对数时间的算法是非常有效的，因为每增加一个输入，其所需要的额外计算时间会变小。

递归地将字符串砍半并且输出是这个类别函数的一个简单例子。它需要O（log n）的时间因为每次输出之前我们都将字符串砍半。 这意味着，如果我们想增加输出的次数，我们需要将字符串长度加倍。

举个例子

```
for(int i = 1; i <= n; i *= 2) 就是o(logn)
```

17.1.3 根号时间

T(n) = O(sqrt(n)

举个例子

```
for(int i = 1; i * i <= n; i++)
```

注意：计算机中一次运算算1的话，1s内大概能运行10^8次运算

17.2 空间复杂度

空间复杂度(Space Complexity)是对一个算法在运行过程中临时占用存储空间大小的量度。同样的用O()来表示

举个例子

```
int a[10000]; //他产生的空间就是4 * 10000字节
```

17.3 练习:

1. 求n以内所有的质数时间复杂度是多少？能不能优化?

2. 二重循环的时间复杂度是多少，三重呢？？

3. 以下代码产生了多少空间大小

```
int a[10000];
double b[100];
float c[101];
```

18. 函数

数学中的函数就是用来代表一个数学式子的。
例如我现在有两个式子 : **y = x + 1，y = x * 2**，我想要问你几个问题。

```
对于式子 y = x + 1，当x = 1的时候答案是多少
对于式子 y = x * 2，当x = 1的时候答案是多少
这样子非常麻烦！
```

假如我这样写 : **y = f(x) = x + 1，y = T(x) = x * 2**，同样的

```
f(1)答案是多少，T(1)答案是多少，非常简单！非常好用！
```

计算机的函数，是一个固定的一个程序段，或称其为一个子程序，它在可以实现固定运算功能的同时，还带有一个入口和一个出口，所谓的入口，就是函数所带的各个参数，我们可以通过这个入口，把函数的参数值代入子程序，供计算机处理；所谓出口，就是指函数的函数值，在计算机求得之后，由此口带回给调用它的程序。
像我们的main，printf其实都是一个函数，只不过C语言作者给我们做好的。所以函数就像我们的工具，我们给他做好了，以后就直接使用就好了
函数的形式是这样的的 带有（一个）参数的函数的声明：

18.1 示例程序

```
#include<stdio.h>

int add(int a,int b){
    printf("我进来了\n");
    return a + b;
}

int main(){
    printf("我在1处\n");
    int c = add(1,2);
    printf("我出来了\n");
    printf("%d",c);
    return 0;
}
```

输出

```
我在1处
我进来了
我出来了
3
```

说明，我们定义的函数，从使用的时候进去，运行完里面的代码后，会回到原处

18.2 函数分类

1. 自定义函数
   类型名标示符+函数名+（类型标示符+参数）{

   // 程序代码

   }

   没有返回值且不带参数的函数的声明：

   void+函数名（）//无类型+函数名{

// 程序代码

}

2. 库函数
像stdio.h就是一个库，printf，scanf都是库函数，后面等学到C++了我们再教大家许多库函数

18.3 一些简单的库函数

18.3.1 示例程序

1. sizeof()

```
#include<stdio.h>

int main(){
    int a = 1;
    double b = 2;
    int c[100];
    printf("%d %d %d",sizeof(a),sizeof(b),sizeof(c));
}
//输出了 4 8 400;
```

第一个要将的函数就是sizeof函数 使用方法如上，得到的答案是占用的数据空间

2. strlen(字符数组地址)

```
#include<stdio.h>
#include<cstring>

int main(){
    char a[10];
    printf("%d",strlen(a));
    a[0] = '1';
    printf("%d",strlen(a));
    a[1] = '1';
    printf("%d",strlen(a));
    a[3] = '1';
    printf("%d",strlen(a));
    printf("%d",strlen(a + 1));
}
//输出了 0 1 3 4 3;
```

大家发现没有，我写了两个头文件。那么 现在使用的函数 strlen就是来自我们库 cstring里面的，他是用来判断从当前位置到最后一个元素字符数组的长度的（即最后一个元素位置 + 1）；

3.strcat()

```
#include<stdio.h>
#include<cstring>

int main(){
    char a[10] = "12";
    char b[10] = "34";
    strcat(a,b);
    printf("%s %s",a,b);
}
//输出了 1234 34
```

这个函数也是cstring里面的是用来拼接字符数组的，把字符数组a和字符数组b的内容拼接起来并且放到字符数组a去

4.strcmp()

```
#include<stdio.h>
#include<cstring>

int main(){
    char a[10] = "12";
    char b[10] = "34";
    char c[10] = "12";
    char d[10] = "56";
    printf("%d ",strcmp(a,b));
    printf("%d ",strcmp(a,c));
    printf("%d",strcmp(d,c));
}
//输出了 -1 0 1;
```

这个函数也是cstring里面的是用来比较字符数组的，把字符数组a和字符数组b的内容进行比较，如果数组a的字典序 > b的字典序那么输出一个负数，如果小于的话输出一个正数，如果等于的话输出0。

5.sqrt()

```
#include<stdio.h>
#include<cmath>

int main(){
    printf("%lf",sqrt(100));
}
//输出了 10.000000;
```

sqrt(a)，返回根号a，记住返回值是浮点数类型

是cmath库里常用的函数

6.pow()

```
#include<stdio.h>
#include<cmath>

int main(){


    printf("%lf %lf",pow(2,3),pow(2,4));
}
//输出了 8.000000 16.000000;
```

pow(a,b) 返回 a的b次幂 a，b可以为整数也可以为浮点数。返回的值是浮点数。
注意：pow(a,-1) 返回的值就是 a分之1

18.4 函数的调用及返回

18.4.1 函数的调用

函数调用一般是这样例如

```
int add(int a,int b){
    return a + b;
}
int main(){
    add(1,2); //函数名字 + 需要传递的参数
    return 0;
}
```

注意，我们的函数底层使用的是数据结构中的栈实现，每次函数调用其实就是一个压入栈的过程，当函数运行结束后，会返回到上一个函数中(主调函数),这个过程我们称之为弹栈。比如说函数A调用函数B，那么程序的控制权由函数 `A` 传递给函数 `B`, 当函数 `B` 运行结束后，会从 函数 `B` 返回到函数 `A`, 下面演示了这个过程。

调用过程

```
操作系统 -> 主函数main() -> A() -> B() -> C() -> D()
```

返回过程

```
操作系统 <- 主函数main() <- A() <- B() <- C() <- D()
```

一句话总结就是,函数从哪儿调用，就会返回到哪儿。(从哪儿来，回哪儿去)，不过值得注意的是: 当函数返回的时候，可以携带结果返回，返回给主调函数。

请看下面这个例子。

自定义函数传递的参数又分为两种：

1. **形参**
   自定义函数中的“形参”全称为"形式参数" 由于它不是实际存在变量，所以又称虚拟变量。实参和形参可以重名。
   大家可以这么理解，由于我们只告诉了函数，我们当前变量的值，所以他只能拷贝一份值去使用，所以是虚拟的，所以称作传值

   ```
   就像
   int b = 1;
   int a = b;
   我们创建了一个变量a去接收b的值，但是我们改变a的值，b的值是不会改变的
   ```

   ```
   int add(int x,int y){
       x = x + y;
       return x;
   }
   int a = 1, b = 1;
      add(a,b);
      printf("%d",a);
      //输出1
   ```

2. **实参**
   实际参数简称“实参”。在调用有参函数时，函数名后面括号中的参数称为“实际参数”，实参可以是常量、变量或表达式。
   大家可以这么理解，相当于给当前变量取了个别名，其他都是一模一样

```
int a = 1;
int &b = a;  //引用
b = 2;
printf("%d",a);
//发现a输出了2
```

```
int add(int &x,int y){
    x = x + y;
    return x;
}

int a = 1, b = 1;
add(a,b);
printf("%d",a);
//输出2
```

18.4.2 函数的返回

函数的返回分为带参数已经不带参数。如果不想带参数就写void 即可

函数的返回是由return来完成的，倘若不写的话，默认执行完返回

```
#include<stdio.h>

int main(){

    printf("123");
}
//输出123 然后结束
```

return还可以用来提前结束函数

```
#include<stdio.h>

int main(){
    return 0;
    printf("123");
}
//什么都没有输出然后结束，说明我们的函数结束与否是由return来控制的
```

18.5练习：

1. 以下代码能否达到预期?

```
void swap(int x,int y){
   int s = x;
   x = y;
   y = s;
}
int main(){
    int a = 1,b = 2;
    swap(a,b);
    //能成功交换a,b的值吗?
}
```

```
void swap(int &x,int &y){
    int s = x;
    x = y;
    y = s;
}
int main(){
    int a = 1,b = 2;
    swap(a,b);
    //能成功交换a,b的值吗？
}
```

```
int cheng(int x,int y){
    return x * y;
}
int main(){
    int a = 1,b = 100;
    a = cheng(1,100);
    //能成功获得值100吗？
}
```

```
int jia(int &x,int &y){
    return x + y;
}
int cheng(int x,int y){
    return x * jia(x + y);
}
int main(){
    int a = 1,b = 100;
    a = cheng(1,100);
    //能成功获得值101吗？
}
```

2.
```
int a = 1;
int &b = a;
int &c = b;
c = 2;
//请问a的值会改变吗？
```

```
int a[10] = {1,2,3};
int &c = a[0];
c = 12;
//请问a[0]的值会改变吗？
```

```
int a[10] = {1,2,3};
int (&c)[10] = a;
c[0] = 10;
printf("%d",a[0]);
//请问合法吗？如果合法输出的值是什么
```

```
int a[10] = {1,2,3};
int &(c[10]) = a;
c[0] = 10;
printf("%d",a[0]);
//请问合法吗？如果合法输出的值是什么
```

3. 思考用函数求得1~100以内的质数

4. 自定义一个比较两个数字的值，返回较大值的函数

5. 思考以下代码能否达到预期值

```
int max2(int x,int y){
    if(x > y) return x;
    return y;
}
//求最大值
```

```
bool isprime(int x){
    for(int i = 2; i * i < x; i++){
        if(x % i == 0) return false;
    }
    return true;
}
//判断是否为质数若是返回true若不是返回false
```

18.6 函数的声明和定义

18.6.1 函数的声明

类似于我们人的目录，告诉我们有哪些东西。

1. 告诉编译器有一个函数叫什么，参数是什么，返回类型是什么。但是具体是不是存在，函数声明决定不了
2. 函数的声明一般出现在函数的使用之前。要满足**先声明后使用**。
3. 函数的声明一般要放在头文件中的

18.6.2 函数的定义

函数的定义是指函数的具体实现，交待函数的功能实现。

18.7 练习：

1. 以下代码能否达到预期?

```
#include<stdio.h>

void jia(int x,int y){
    return cheng(1,2);
}
void cheng(int x,int y){
    return x * y;
}
```

```
#include<stdio.h>
int jia(int x,int y);
int cheng(int x,int y);

int jia(int x,int y){
    return cheng(1,2);
}
int cheng(int x,int y){
    return x * y;
}
```

```
#include<stdio.h>
int jia(int x,int y);
void jia(int x,int y);
```

```
#include<stdio.h>
int jia(int x,int y,int z);
int jia(int x,int y);
```

```
#include<stdio.h>
int jia(int x,int y);
int jia(int x,char y);
```

```
#include<stdio.h>
int a;
int jia(int a,int b){
    printf("%d",a);
}
```

2. 请问函数里面的变量作用域和生命周期是?

3. 函数的作用域和生命周期是?

18.8 总结:

1. 函数是一个把行为封装起来的行为
2. 函数调用其他函数后，会保存原来的状态，直到返回。
3. 函数定义和函数声明最好分开写。
4. 函数名字可以取相同的名字，但是传递的变量需要有所不同，要么个数不同，要么变量类型不同，但是返回类型不同不行。
5. 函数可以把我们的代码很好的模块化，更简洁，更好看

19. 函数递归

19.1 什么是递归?

程序调用自身的编程技巧称为递归（ recursion）。

举个例子

▼ 和尚讲故事
从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？“从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？‘从前有座山，山里有座庙，庙里有个老和

尚，正在给小和尚讲故事呢！故事是什么呢？……”’

递归做为一种算法在程序设计语言中广泛应用。 一个过程或函数在其定义或说明中有直接或间接

调用自身的

一种方法，它通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，

递归策略

只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。

递归的主要思考方式在于：把大事化小

```
//最简单的递归函数
#includ<stdio.h>

int main(){

    printf("hello\n");
    main();
    return 0;
}
//一直输出hello说明，我们需要暂停一个条件去暂停它
```

19.2 递归的必要条件

- 存在限制条件，当满足这个限制条件的时候，递归便不再继续。
- 每次递归调用之后越来越接近这个限制条件

```
//最简单的递归函数
#includ<stdio.h>

int i = 1;
int main(){
    if(i <= 4) return 0;
    i++;
    printf("hello\n");
    main();
    return 0;
}
//一直输出hello说明，我们需要暂停一个条件去暂停它
```

举个最简单的递归例子：接受一个整型值（无符号），按照顺序打印它的每一位。

例如：

输入：1234，输出 1 2 3 4

想要打印 1 2 3 4，我们先思考，如何把他的每一位都拿出来

拿出最后一位是 % 10

拿出倒数第二位是 / 10 % 10

拿出倒数第三位是 / 10 / 10 % 10

我们发现我们一直在重复一个动作 / 10 % 10

直到我们的x为0

```
void get(int x){
    if(x == 0) return;
    printf("%d ",x % 10);
    get(x / 10);
}
//如果这样写的话输出 4 3 2 1
//那么我们想要输出1 2 3 4，只要改变输出时机就行了
void get(int x){
    if(x == 0) return;
    get(x / 10);
    printf("%d ",x % 10);
}
//如果是这样子的话，先把每一位都拿了，最后再输出。
```

我们发现在调用自己之前写的代码，按照调用函数的顺序执行，在调用之后写的代码，按照调用函数的反顺序执行。

19.3 练习：

1. 求n的阶乘

2. 求第n个斐波那契数。（不考虑溢出） f[1] = 1,f[2] = 1,f[n] = f[n - 1] + f[n - 2]

3. 给出一个整数n表示我要走到第n个楼梯的方案数，我从第0个楼梯开始走，每次可以向上走一个或者两个

19.4 递归和循环对比

- 递归速度慢,消耗内存高，循环速度快，消耗内存低
- 无限递归会导致系统崩溃，而无限循环会消耗 `CPU` 周期
- 递归使代码更小巧，而迭代使代码更长。
- 函数递归可以自己决定干事情的顺序，在调用下一个函数之前干事情，那么干事情的顺序就是调用顺序，如果在调用下一个函数之后去干事情，那么干事情的顺序就是调用顺序的反顺序
- 函数的空间都是独立，变量之间也是独立的，函数之间可以传递参数，函数之间也可以返回值

20. 结构体

有时候要大量存储批量数据，比如说某位考生的信息，可以考虑使用数组，但是数组只能存储一组同样数据类型的信息，如果同时记录考生的信息，成绩等不同的信息就不能使用一个数组来存储了。字符串可以用来存储很多种类信息，但是如果存储数字的话，我们要一个个分隔开也很麻烦，所以这里介绍一个方法叫做结构体。C语言的结构体是由一系列具有相同类型或不同类型的数据构成的数据集合。

```
struct 名字1{ //大括号里面包含数据类型
    数据类型1 成员变量1
    数据类型2 成员变量2
}[结构体变量名];

struct 名字1 名字2;
```

假设我现在需要存储学生的名字，数学，语文，英语成绩，我可以这么写

```
struct Student{
    char Name[10];
    int Math;
    int Chinese;
    int English;
};
Student student;
//想要使用学生的名字就是 student.name;
//想要使用学生的数学成绩就是 student.math
```

现在我有 两名学生：
Steven 他的数学语文英语成绩分别为， 100 ， 90，80。
Hellen 他的数学语文英语成绩分别为， 80 ， 95，90。
Haohao 他的数学语文英语成绩分别为， 95 ， 90，93。

```
//我可以这么存
struct Student{
    char Name[10];
    int Math;
    int Chinese;
    int English;
}student[2];
student[0] = {"Steven",100,90,80};
student[1].Name = "Hellen";
student[1].Math = 80;
student[1].Chinese = 80;
student[1].English = 80;
student[2] = Student{"Haohao",95,90,93};
```

21. 指针

21.1 普通指针

指针，是C语言中的一个重要概念及其特点，也是掌握C语言比较困难的部分。指针也就是内存地址，指针变量是用来存放内存地址的变量，在同一CPU构架下，不同类型的指针变量所占用的存储单元长度是相同的，而存放数据的变量因数据的类型不同，所占用的存储空间长度也不同。有了指针以后，不仅可以对数据本身，也可以对存储数据的变量地址进行操作。

指针描述了数据在内存中的位置，标示了一个占据存储空间的实体，在这一段空间起始位置的相对距离值。在C/C++语言中，指针一般被认为是指针变量，指针变量的内容存储的是其指向的对象的首地址，指向的对象可以是变量（指针变量也是变量），数组，函数等占据存储空间的实体。

众所周知 &a指的是a的地址那么让我们试验一下

```
int b = 1;
int a = &b; //这样子可以写吗?
```

我们发现是过不了编译的，因为我们的内存地址只能用我们的指针来存储

```
int a = 1;
int *p = &a; //我们念作p是一个指向整形类型的指针，指向的是整形变量a的地址。
printf("%d %d",p,*p); //如果直接输出p的话就是输出a的地址，*p能输出该地址存储的内容，*也称作解地址符号
```

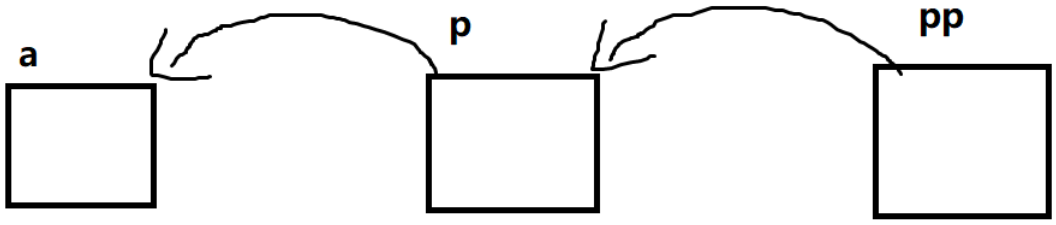
当我们更改指针的值得时候我们同时也能够更改指向那个地址的值

```
int a = 1;
int *p = &a;
*p = 3;
printf("%d",a);
```

当然我们的指针也有地址，所以我们也可以使用一个指针去指向我们的指针。

```
int a = 1;
int *p = &a;
int *pp = p;//因为我们的p指向的就是a的地址，所以pp也可以直接指向
printf("%d",*pp); //输出1
```

int a；int *p = &a; intpp= &p; 如图**



说明p指向了a的地址，那么p == &a 两边同时添上一个*变成

***p == *(&a) == a**

pp == &p 那么 *pp == *(&p) == &a

那么pp == **(&p)==(*&a) == a**

注意：以下写法

```
int a = 1;
int *p = &a;
int **pp = &p; //他这样叫，pp是一个指针指向的是 一个指向整形变量地址的指针的地址，所以他必须指向一个指针的地址。
//不能这么写 int **pp = p;
printf("%d",**pp);//输出1
```

注意：指针不能直接指向一块地址例如：

```
int *p = 1;
//这是禁止的，因为我们的空间也会保存我们操作系统的一些东西，如果我们直接能够使用地址去更改的话，会导致操作系统出错
//所以在操作指针的时候需要注意
```

21.2 练习：

1. 以下操作合法吗?若合法输出的结果是什么?

```
int a[100];
a[0] = 1;
int *p = a;
printf("%d",*p);
```

```
int a[100];
a[0] = 1,a[1] = 2;
int *p = a;
printf("%d",*(p + 1));
```

```
int a = 10;
int *p = &a; //一级指针
int **pp = &p;
int ***ppp = &pp;
printf("%d",***ppp);
```

```
//倘若a的地址为0 p的地址为 10，pp的地址为100，ppp的地址为1000
int a = 10;
int *p = &a;
int **pp = &p;
int ***ppp = &pp;
printf("%d",***ppp);
printf("%d",**ppp);
printf("%d",*ppp);
printf("%d",ppp);
printf("%d\n",&ppp);
printf("%d\n",&pp);
printf("%d\n",&p);
printf("%d\n",&a);
```

```
int a = 100;
int *p = &a;
int **pp = &p;
int *c = &pp;
printf("%d",*c);
```

21.3 数组指针，指针数组

大家看名字可能看不出什么区别，但是在他们名字中间加个的就很明确了

21.3.1 数组的指针

说明他是一个指针，什么指针，指向数组的指针

```
int (*p)[10];//这个就读作p是一个指针指向一个大小为10的数组的首地址
int a[10] = {1};
p = &a;
printf("%d",(*p)[0]);
```

21.3.2 指针的数组

说明他是一个数组，什么数组，里面都是指针的数组

```
int *(p[10]);
int a = 1;
p[0] = &a;
printf("%d",*(p[0]));
```

那么如果我们不写括号的话是什么呢?

```
int *p[10];//他先和数组结合，所以他是指针数组
```

22. C语言中其他关键词

22.1 define宏定义

宏是在程序运行前的一个准备工作

22.1.1 无参宏

一种最简单的宏的形式如下：

> 一种最简单的宏的形式如下：
> #define 宏名 替换文本每个#define行（即逻辑行）由三部分组成：第一部分是指令 #define 自身，“#”表示这是一条预处理命令，“define”为宏命令。第二部分为宏（macro），一般为缩略语，其名称（宏名）一般大写，而且不能有空格，遵循C变量命令规则。“替换文本”可以是任意常数、表达式、字符串等。在预处理工作过程中，代码中所有出现的“宏名”，都会被“替换文本”替换。这个替换的过程被称为“宏代换”或“宏展开”（macro expansion）。“宏代换”是由预处理程序自动完成的。在C语言中，“宏”分为两种：无参数 和 有参数。

无参宏是指宏名之后不带参数，上面最简单的宏就是无参宏。

```
#define M 5                 // 宏定义
#define PI 3.14             //宏定义
int a[M];                   // 会被替换为： int a[5];
int b = M;                  // 会被替换为： int b = 5;
printf("PI = %.2f\n", PI); // 输出结果为： PI = 3.14
```

注意宏不是语句，结尾不需要加“;”，否则会被替换进程序中，如：

```
#define N 10;                // 宏定义
int c[N];                    // 会被替换为： int c[10;];
//error:… main.c:133:11: Expected ']'
```

如果要写宏不止一行，则在结尾加反斜线符号使得多行能连接上，如：

```
#define HELLO "hello \
the world"
```

注意第二行要对齐，否则，如：

```
#define HELLO "hello the wo\
  rld"
printf("HELLO is %s\n", HELLO);
//输出结果为： HELLO is hello the wo  rld
```

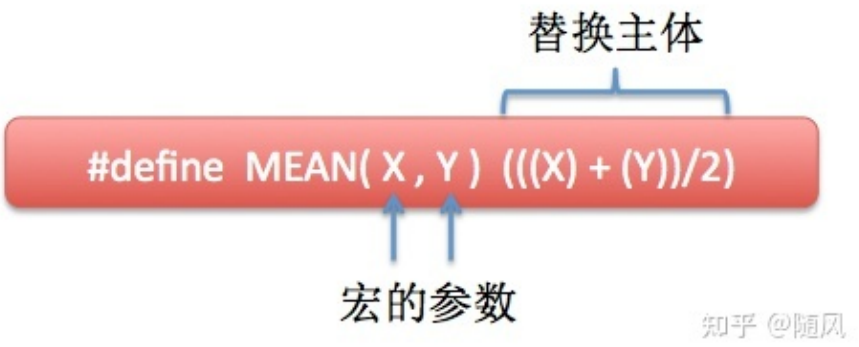
也就是行与行之间的空格也会被作为替换文本的一部分

而且由这个例子也可以看出：宏名如果出现在源程序中的“”内，则不会被当做宏来进行宏代换。

宏可以嵌套，但不参与运算：

```
#define M 5                  // 宏定义
#define MM M * M             // 宏的嵌套
printf("MM = %d\n", MM);     // MM 被替换为: MM = M * M, 然后又变成 MM = 5 * 5
```

22.1.2 有参宏



宏调用：

宏名（实参表）；

printf(“MEAN = %d\n”, MEAN(7, 9)); // 输出结果： MEAN = 8

和函数类似，在宏定义中的参数成为形式参数，在宏调用中的参数成为实际参数。

而且和无参宏不同的一点是，有参宏在调用中，不仅要进行宏展开，而且还要用实参去替换形参。如：

```
#define M 5                             //无参宏
#define COUNT(M) M * M                  //有参宏
printf("COUNT = %d\n", COUNT(10));      // 替换为： COUNT(10) = 10 * 10
                                        // 输出结果： COUNT = 100
```

这看上去用法与函数调用类似，但实际上是有很大差别的。如：

```
#define COUNT(M) M * M                  //定义有参宏
int x = 6;
printf("COUNT = %d\n", COUNT(x + 1));// 输出结果： COUNT = 13
printf("COUNT = %d\n", COUNT(++x));  // 输出结果： COUNT = 56
```

这两个结果和调用函数的方法的结果差别很大，因为如果是像函数那样的话，COUNT(x + 1)应该相当于COUNT(7)，结果应该是 7 * 7 = 49，但输出结果却是21。原因在于，预处理器不进行技术，只是进行字符串替换，而且也不会自动加上括号（），所以COUNT(x + 1)被替换为 COUNT(x + 1 * x + 1)，代入 x = 6，即为 6 + 1 * 6 + 1 = 13。而解决办法则是：尽量用括号把整个替换文本及其中的每个参数括起来：

```
#define COUNT(M) ((M) * (M))
```

但即使用括号，也不能解决上面例子的最后一个情况，COUNT(++x) 被替换为 ++x * ++x，即为 7 * 8 = 56，而不是想要 7 * 7 = 49，解决办法最简单的是：不要在有参宏用使用到“++”、“–”等。

22.2 练习：

1. 以下代码输出什么？

```
#define x 5 + 1
printf("%d",x * x);
```

```
#define x (5 + 1)
printf("%d",x * x);
```

```
#define add(x,y) x + y
printf("%d\n",add(1,2));
```

22.3 typedef

typedef是在计算机编程语言中用来为复杂的声明定义简单的别名，它与宏定义有些差异。它本身是一种存储类的关键字，与auto、extern、mutable、static、register等关键字不能出现在同一个表达式中。

我们一般用来给我们的变量取别名例如：

```
typedef long long ll;
typedef double db;

ll a;db b;
```

当然你也可以使用宏定义

```
#define ll long long
#define db double
```

22.4 const

const是一个C语言（ANSI C）的关键字，具有着举足轻重的地位。它限定一个变量不允许被改变，产生静态作用。使用const在一定程度上可以提高程序的安全性和可靠性。另外，在观看别人代码的时候，清晰理解const所起的作用，对理解对方的程序也有一定帮助
因为我们的变量都是可变的，所以也可以有常量，不能更改的变量
使用方法

```
const int a = 1000;
```

但是以下方法不行

```
const int a;
a = 1000;//后面赋值就错误
```

常量常常用来定义全局变量

以下写法编译是过不了的，因为全局变量的数组大小只能用常量定义

```
#include<stdio.h>
int a = 100;
int b[a];
int main(){
    return 0;
}
```

以下正确

```
#include<stdio.h>
const int a = 100;
int b[a];
int main(){
    return 0;
}
```

当然常量也可以用来定义指针

```
int a = 10;
const int *p = &a;
```

但是大家觉得以下的代码正确吗?

```
int a = 10;
const int *p = &a;
a = 100;
printf("%d",*p);
```

我们发现编译成功了，并且*p的值输出了100

但是我们直接修改*p是不可以的，所以指针常量的值还是可能会改变的

还有一个要注意的点:你们觉得以下代码对吗?

```
int a = 1,b = 2;
const int *p = &a;
p = &b;
printf("%d\n",*p);
```

我们发现是可以编译的，并且成功输出了2

所以我们发现 const int *p; 仅仅是不能改变 *p的值，但是p指向的地址可以改变。

所以我们想要 *p和地址都不改变我们需要这样子写

```
int a = 1,b = 2;
const int * const p = &a;
p = &b;
printf("%d\n",*p);
```

22.4 signed和unsigned

这两个个关键字一般是配合我们的变量使用的

例如 signed int，signed char，unsigned int，unsigned char来使用。

signed的意思是有符号的意思也就是正数负数的意思，我们一般的 int，char他都是属于有符号数，所以我们就算不加signed他默认为有符号数，也就是说 signed int 和 int是一个东西。

unsigned的医生就是无符号的意思也就是说没有负数的意思，我们如果给变量加上了unsiged他就从补码变成了无符号补码，无符号的补码不管哪一位都是正数。也就是说补码的最高位也变成了正数。那么unsiged int的范围就变成了 0 ~ 2^{32}-1，当无符号数溢出的时候相当于减去了一个2^{32}